VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Samoopravné kódy a jejich aplikace

Error detecting and error correcting codes and theirs applications

2018 Jakub Salamon

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Zadání bakalářské práce

Student: **Jakub Salamon**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 1103R031 Výpočetní matematika

Téma: Samoopravné kódy a jejich aplikace

Error detecting and error correcting codes and theirs applications

Jazyk vypracování: čeština

Zásady pro vypracování:

Samoopravné kódy jsou dnes používány téměř ve všech moderních digitálních elektronických systémech a datových sítích. Teorie kódování spojuje několik disciplín, především matematiku s informatikou a elektrotechnikou. Mezi populárními aplikacemi bychom mohli zmínit např. kódovací techniky pro přenos informací v mobilních sítích jako je global system for mobile comunications (GSM), nebo pro High-Definition TeleVision (HDTV) systémy, pro Compact Disc (CD) a Digital Video Disc (DVD). V běžném životě se často setkáváme s řadou kódů schopných detekovat chyby jako: ISBN, rodná čísla, čísla účtů atd. Teorie kódování je však popsána v abstraktním jazyce specialistů matematických oborů, který ale je hůře srozumitelný pro studenty inženýrských oborů informatiky a elektrotechniky.

Účelem této práce je na příkladech demonstrovat potřebné matematické aspekty vybrané skupiny samoopravných kódů v návaznosti na jejich aplikace. Přitom důraz bude kladen na srozumitelnost a názornost přičemž se pokusíme vyhnout vysoce teoretickým obecným algebraickým konstrukcím.

Práci lze rozdělit do následujících částí:
- úvod do problematiky samoopravných kódů,
- nezbytný teoretický matematický základ potřebný pro práci se zvolenou skupinou samoopravných kódů, vybrané kapitoly z algebry,
- prezentace a konstrukce vybraných kódů vždy v souvislosti s konkrétní aplikací,
- rozbor vlastností a možností vybrané skupiny kódů.

Seznam doporučené odborné literatury:

- R. Hill: A First Course in Coding Theory, Clarendon Press, Oxford, (1986), ISBN 0-19-853803-0.

- I. S. Reed, X. Chen: Error - Control Coding for Data Networks, Kluwer Academic Publishers, (2001), ISBN-0-7923-8528-4.

- Steven Roman: Introduction to Coding and Information Theory, Springer - Verlag, New York, (1997), ISBN 0-387-94704-3.

- odborné články a další literatura dle pokynů vedoucího

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Tereza Kovářová, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. RNDr. Jiří Bouchala, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 ..... Šalamon .....

Na tomto místě bych rád poděkoval vedoucí práce paní Mgr. Tereze Kovářové, Ph.D za ochotu, pomoc, cenné rady, připomínky a čas, který mně a této práci věnovala.

Abstrakt

Samoopravné kódy jsou prostředkem k opravování chyb, které nastávají při přenosu dat komunikačním kanálem. Tato práce se věnuje skupině blokových kódů, které jsou lineární a jejím dvěma podtřídám, cyklickým a Reed-Solomonovým kódům. Reed-Solomonovy kódy jsou v dnešní době poměrně hojně využívány, a proto jsou zvoleny jako hlavní předmět této práce. Uvedeny jsou způsoby konstrukcí, kódování a dekódování zmíněných blokových kódů a k tomu nezbytný matematický aparát. Kapitola o Reed-Solomonových kódech navíc zmiňuje konkrétní aplikace. Cílem je přinést srozumitelný a ucelený souhrn poznatků o lineárních a cyklických kódech se zaměřením na Reed-Solomonovy kódy. Konstrukce, kódování a dekódování jsou demonstrovány na příkladech.

Klíčová slova: samoopravné kódy, blokové kódy, lineární kódy, cyklické kódy, Reed-Solomonovy kódy, konečná tělesa, kódování, dekódování, hlavní problém teorie kódování

Abstract

Error-correcting codes act as a means to correct errors which occur during data transfer through a communication channel. This thesis is devoted to a group of block codes that are linear and its two subclasses, cyclic and Reed-Solomon codes. In this day and age the Reed-Solomon codes are being used abundantly and for that reason were chosen as a main topic for this thesis. Principles of construction, coding and decoding of mentioned block codes are presented including necessary mathematical background. The chapter about Reed-Solomon codes also includes specific applications. The goal of this thesis is to bring a comprehensive summary of linear block codes with emphasis on Reed-Solomon codes. Constructions, coding and decoding are demonstrated on examples

Key Words: error-correcting codes, block codes, linear codes, cyclic codes, Reed-Solomon codes, finite fields, coding, decoding, main coding theory problem

Obsah

Seznam použitých zkratek a symbolů

RS	–	Reed-Solomonův
MDS	–	Maximum Distance Separable
GF	–	Galois Field
CD	–	Compact Disc
DVD	–	Digital Versatile Disc
BD	–	Blu-Ray Disc
CCSDS	–	Consulative Committee for Space Data Systems
CIRC	–	Cross-Interleaved Reed-Solomon Coding
QR	–	Quick Response

Seznam obrázků

Seznam tabulek

1 Úvod

V dnešní digitální době je potřeba, abychom veškerá data mohli přenášet co možná nejspolehlivěji a nejrychleji. Zároveň je ale prakticky nemožné úplně zamezit chybám, které při přenosu nastávají, ať už jsou způsobeny vnějšími vlivy nebo samotným přenosovým médiem. Chybná data ztrácejí na hodnotě a opakovaný přenos téže informace, je-li vůbec možný, zbytečně zvyšuje provozní náklady. Jedním z prostředků, které tyto problémy dokáží do jisté míry řešit, jsou samoopravné kódy. Druhů samoopravných kódů je celá řada. Vybrali jsme skupinu blokových kódů, které nesou název Reed-Solomonovy kódy. Tato třída je zajímavá zejména z pohledu širokého uplatnění a existence efektivních kódovacích a dekódovacích algoritmů.

V této práci nejprve zavedeme matematický aparát, který je potřebný k porozumění, popisu a práci s blokovými samoopravnými kódy. Protože jsou Reed-Solomonovy kódy konstruovány s využitím aritmetiky konečných těles, ukážeme si, jak s touto základní algebraickou strukturou pracovat. Následně definujeme pojem samoopravný kód jako takový a budeme se zabývat tzv. hlavním problémem teorie kódování. Poté přejdeme k základní třídě blokových samoopravných kódů, lineárním kódům. Kromě definice, výhod a nevýhod, které tato třída přináší, popíšeme také kódovací a dekódovací proces a vše ukážeme na příkladech. Podtřídou lineárních kódů, kterou se budeme zabývat dále, jsou pak cyklické kódy. Srovnáme je s lineárními kódy, opět uvedeme i kódovací a dekódovací algoritmy a nastíníme způsob implementace kodérů či dekodérů pomocí posuvných registrů. Obě zmíněné třídy jsou nezbytné k porozumění Reed-Solomonovým kódům. V kapitole věnované Reed-Solomonovým kódům uvedeme dva typy konstrukce, způsob kódování s využitím vlastností cyklických kódů a rozebereme postupy dekódování. Na závěr zmíníme některé z mnoha konkrétních aplikací, jež Reed-Solomonovy kódy v dnešní době mají.

Cílem práce je čtenáře ve srozumitelné formě seznámit s problematikou samoopravných kódů se zaměřením na Reed-Solomonovy kódy. Za tímto účelem uvedeme řadu příkladů k demonstraci kódovacích a dekódovacích algoritmů, dostupných z hlediska složitosti potřebného matematického aparátu. Tato práce může sloužit i jako jakýsi "rozcestník", který čtenáři pomocí uvedených zdrojů umožní dále prohlubovat znalosti v oblasti především blokových samoopravných kódů.

2 Základní pojmy

V této kapitole zavedeme základní pojmy a matematický aparát, jež je nutný k porozumění problematiky samoopravných kódů, konkrétně lineárních, cyklických a posléze Reed-Solomonových.

2.1 Konečné těleso

Pro definici konečného tělesa je nejprve nutné nadefinovat pojmy, na kterých budeme stavět. Samotné definice a vlastnosti jsou inspirovány tituly [1], [2] a také vlastními zápisky z přednášek kurzu Algebra. Zároveň na titul [1] odkazuji čtenáře i pro vysvětlení některých pojmů abstraktní algebry, jejichž znalost následující text předpokládá. Konkrétně jsou to pojmy *grupa, pologrupa, neutrální prvek, komutativita aj.*

Definice 2.1.1 *Mějme neprázdnou množinu R a na ní dvě binární operace, které označme například "+"a "$\cdot$". Okruhem nazvme uspořádanou trojici $(R,+,\cdot)$ právě tehdy, když platí:*

- *$(R,+)$ je **komutativní** grupa*
- *$(R,\cdot)$ je pologrupa*
- *Oboustranná distributivita násobení ke sčítání: $\forall a,b,c \in R : a \cdot (b+c) = a \cdot b + a \cdot c \wedge (b+c) \cdot a = b \cdot a + c \cdot a$*

Poznámka Operace + a $\cdot$ nemusí nutně představovat sčítání a násobení (i když většinou představují). Jde pouze o označení dvou binárních operací, které splňují dané vlastnosti.

Definice 2.1.2 *Okruh $(R,+,\cdot)$ nazveme komutativní, pokud je operace $\cdot$ komutativní.*

Definice 2.1.3 *Komutativní okruh $(R,+,\cdot)$ nazveme těleso, jestliže $(R\backslash\{0\},\cdot)$ je grupa.*

Poznámka Uvedené označení 0 nemusí být konkrétně číslovka nula. Jedná se o neutrální prvek grupy (R,+).

Definice 2.1.4 *Těleso $(R,+,\cdot)$ nazveme konečné těleso, pokud má konečný počet prvků.*

Příklad 2.1.5 Jako příklad tělesa uveďme množinu $\mathbb{Z}_3 = \{0,1,2\}$, kde operace sčítání a násobení jsou prováděny modulo 3 (tzn. počítá se se zbytkovými třídami po dělení 3). Uspořádané dvojice $(\mathbb{Z}_3,+)$ a $(\mathbb{Z}_3\backslash\{0\},\cdot)$ tvoří grupy. Jedná se o konečné těleso $(\mathbb{Z}_3,+,\cdot)$. Tabulky výsledků pro obě operace vypadají následovně:

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Tabulka 1: Sčítání modulo 3

·	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Tabulka 2: Násobení modulo 3

Konečná tělesa se obyvkle značí **GF(q)**, kde q představuje počet prvků. Označení GF vychází z alternativního pojmenování konečných těles, konkrétně **Galoisovo těleso** (anglicky *Galois Field*). Pojmenováno je podle francouzského matematika *Évariste Galoise* (1811-1832), jež je považován za zakladatele teorie grup. Těleso $(\mathbb{Z}_3, +, \cdot)$ z příkladu **2.1.5** můžeme označit jako GF(3).

2.2 Vlastnosti a konstrukce konečných těles

V této podkapitole se budeme věnovat konstrukci konečných těles a uvedeme také důležité vlastnosti, které v pozdějších kapitolách využijeme při konstrukci cyklických a RS-kódů.

V konečném tělese GF(3) z příkladu **2.1.5** je patrné, že pokud vezmeme prvek 1 a přičteme k němu 1, pochopitelně dostaneme 2. Pokud však vezmeme 2 a přičteme 1, dostaneme 0, protože počítáme modulo 3.

Jelikož se jedná o těleso, máme jistotu, že se v něm nachází prvek 1, reprezentující neutrální prvek pro druhou operaci. Ve většině případů tomu tak je, nicméně nemusí se vždy jednat přímo o číslovku jedna. Budeme-li na něj neustále aplikovat první operaci (v našem případě sčítání), kvůli konečnosti nemůžeme pořád dostávat nové prvky. V našem příkladě používáme modulární aritmetiku, a tudíž $1 + 1 = 2$, ovšem $1 + 1 + 1 = 0$. Po určitém počtu sečtení jsme se vrátili zpět na nulu. Takováto situace musí nastat obecně v každém konečném tělese.

Definice 2.2.1 *Mějme konečné těleso GF(q) a v něm prvek 1. Pak nejmenší přirozené číslo p takové, že 1+1+...+1 (p krát) = 0, se nazývá charakteristika tělesa.*

Charakteristika tělesa GF(3) z příkladu **2.1.5** je $p = 3$.

Věta 2.2.2 [2] *Charakteristika p konečného tělesa je prvočíslem.*

Důkaz Předpokládejme, že charakteristika p není prvočíslem a můžeme ji tedy zapsat jako součin dvou **menších** čísel *a* a *b*. Z definice charakteristiky však víme, že

$$1+1+...+1 \text{ (p krát)} = 0.$$

Pokud je však p=ab, dostáváme

$$[1+1+...+1 \text{ (a krát)}] \cdot [1+1+...+1 \text{ (b krát)}] = 0.$$

Odtud je patrné, že buď 1+1+...+1 (a krát) = 0 nebo 1+1+...+1 (b krát) = 0. Toto je však v rozporu s definicí charakteristiky, jelikož p má být **nejmenší** přirozené číslo, které splňuje danou podmínku. Z toho jasně plyne, že p musí být prvočíslem. □

Věta 2.2.3 [1] *Pro konečné těleso GF(q) s charakteristikou p platí, že pokud $q \neq p$ pak $q = p^m$, pro $m \in \mathbb{N}$.*

Důkaz Ke korektnímu důkazu bychom potřebovali zavést pojmy, týkající se vektorových prostorů a dalších vlastností konečných těles. Základní myšlenku důkazu ovšem uvedeme, jelikož nám umožní na konečná tělesa nahlédnout jako na vektorový prostor. Pro přesné definice pojmů, jako například *vektorový prostor, báze, rozšíření tělesa aj.* odkážeme čtenáře na zdroj [1].

Je-li charakteristika tělesa GF(q) rovna p, těleso obsahuje podtěleso (označme například K), které je izomorfní s tělesem $(\mathbb{Z}_p, +, \cdot)$. Podtěleso K bude obsahovat p prvků. Dále těleso GF(q) můžeme chápat jako rozšíření tělesa K, a tudíž GF(q) je lineární prostor nad tělesem K. Jelikož je těleso GF(q) konečné, jeho dimenze musí být také konečná. Dimenzi označme m. Pokud je dimenze GF(q) rovna m, jeho báze B bude obsahovat m prvků, $B = [\alpha_1, \alpha_2, ..., \alpha_m]$. Každý vektor tohoto lineárního prostoru lze proto reprezentovat jako lineární kombinaci prvků báze $\alpha_1, \alpha_2, ..., \alpha_m$. Každý prvek $\alpha \in GF(q)$ je reprezentován jako:

$$\alpha = b_1\alpha_1 + b_2\alpha_2 + ... + b_m\alpha_m, \text{ kde } b_i \in K\ (i = 1, ..., m).$$

Pro každé $b_i \in K$ máme p možností, jakých hodnot může nabývat. Celkový počet různých možností, jak sestavit prvek α je p^m. Všechny tyto prvky tvoří celé těleso GF(q), a proto počet prvků q tělesa GF(q) je p^m. □

Příklad 2.2.4 Uspořádaná trojice $(\mathbb{Z}_4, +, \cdot)$ netvoří těleso, protože $(\mathbb{Z}_4 \backslash \{0\}, \cdot)$ není grupa. To však neznamená, že neexistuje konečné těleso GF(4). Namísto množiny zbytkových tříd modulo 4 mějme množinu $\mathbb{Z}_2[x]$ tvořenou polynomy nejvýše prvního stupně, kde koeficienty mohou nabývat pouze hodnot 0 nebo 1. Taková množina je tvořena prvky $\{0, 1, x, x + 1\}$. Operace + a $\cdot$ budou prováděny modulo ireducibilní (dále nedělitelný) polynom $x^2 + x + 1$ stupně 2. Uveďme tabulky obou operací, které umožní čtenáři snadno ověřit, že uspořádaná trojice $(\mathbb{Z}_2[x], +, \cdot)$ konečné těleso tvoří, a můžeme ji proto označit jako GF(4).

+	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

Tabulka 3: Sčítání v GF(4)

·	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

Tabulka 4: Násobení v GF(4)

Poznámka Pokud máme konečná tělesa GF(q), kde q je rovno prvočíslu p, vždy můžeme používat aritmetiku modulo p. V případě, že je počet prvků GF(q) roven mocnině prvočísla, a tudíž $q = p^m$, je příkladem těleso polynomů $\mathbb{Z}_p[x]$, kde koeficienty polynomů jsou ze $\mathbb{Z}_p$ a operace jsou prováděny modulo ireducibilní polynom stupně m. V minulém příkladě **2.2.4** jsme měli těleso GF(4)=GF(2^2), tvořené polynomy nad $\mathbb{Z}_2$ modulo ireducibilní polynom stupně 2. Takováto tělesa budeme využívat při konstrukcích, kódování i dekódování.

Velmi podobně jako charakteristiku získáme i tzv. **řád prvku**. Mějme nějaký nenulový prvek a z GF(q). Protože pro všechny nenulové prvky z GF(q) platí uzavřenost k násobení, musí být všechny mocniny tohoto prvku (mocniny rozumějme jako $a^1 = a, a^2 = a \cdot a, a^3 = a \cdot a \cdot a$) taktéž nenulové prvky GF(q). A protože se jedná o těleso s konečným počtem prvků, musí docházet k opakování. Máme tedy dvě různé mocniny k a m, pro které platí, že k>m a přitom $a^k = a^m$. Pokud tuto rovnici přenásobíme a^{-k}(inverzí k a^k), dostaneme $1 = a^{m-k}$. Z toho jasně plyne, že existuje nejmenší přirozené číslo n takové, že $a^n = 1$.

Definice 2.2.5 *Mějme konečné těleso GF(q) a v něm libovolný nenulový prvek a. Nejmenší přirozené číslo n takové, že* $a^n = 1$ *, se nazývá řád prvku.*

Prvek 2 tělesa GF(3) z příkladu **2.1.5** je řádu $n = 2$, protože v GF(3) platí, že $2^2 = 2 \cdot 2 = 1$. Prvek x z příkladu **2.2.4** je řádu $n = 3$, protože v GF(4) je $x^3 = x \cdot x \cdot x = 1$.

Následující dvě věty uvádějí užitečné vlastnosti prvků konečných těles, usnadňující aritmetiku s těmito prvky.

Věta 2.2.6 [2] *Nechť a je nenulový prvek GF(q). Pak* $a^{q-1} = 1$.

Důkaz Mějme všechny nenulové prvky z GF(q), které označme $b_1, b_2, ..., b_{q-1}$. Zřejmě také prvky $a \cdot b_1, a \cdot b_2, ..., a \cdot b_{q-1}$ jsou nenulové a různé, a proto máme

$$(a \cdot b_1) \cdot (a \cdot b_2) \cdots (a \cdot b_{q-1}) = b_1 \cdot b_2 \cdots b_{q-1},$$
$$a^{q-1} \cdot (b_1 \cdot b_2 \cdots b_{q-1}) = b_1 \cdot b_2 \cdots b_{q-1}.$$

A protože $a \neq 0$, a také $(b_1 \cdot b_2 \cdots b_{q-1}) \neq 0$, musí platit, že $a^{q-1} = 1$. □

Věta 2.2.7 [2] *Nechť a je nenulový prvek z GF(q) a nechť n je řád prvku a. Pak n dělí* $q-1$.

Důkaz Předpokládejme, že n nedělí $q-1$. Pak po vydělení $q-1$ číslem n dostáváme

$$q-1 = kn + r, \text{ kde } 0 < \text{r} < \text{n}.$$

Pak máme

$$a^{q-1} = a^{kn+r} = a^{kn} \cdot a^r = (a^n)^k \cdot a^r.$$

A protože $a^{q-1} = 1$ a také $a^n = 1$, dostáváme, že $a^r = 1$, což je nemožné, protože $0 < r < n$. Tím pádem musí být $r = 0$ a n musí dělit $q-1$. □

Z předchozí věty lze odvodit, že například v tělese GF(3) najdeme pouze prvky řádu 1 nebo 2. V tělese GF(5) zase prvky řádu 1,2,4 a v tělese GF(8) potom prvky řádu 1 a 7.

Definice 2.2.8 *Mějme konečné těleso GF(q). Prvek z GF(q), jehož řád je roven* $q-1$*, se nazývá primitivní.*

Důležitou skutečností je, že postupným umocňováním primitvního prvku jsme schopni "vygenerovat" všechny nenulové prvky GF(q).

Primitivním prvkem tělesa GF(3) z příkladu **2.1.5** je prvek 2. Těleso GF(4) z příkladu **2.2.4** má dva primitivní prvky x a $x+1$.

Věta 2.2.9 [3] *Každé konečné těleso GF(q) obsahuje primitivní prvek.*

Důkaz Nejprve ukážeme, že množina všech nenulových prvků konečného tělesa (množinu označme R\{0}) s druhou operací (označenou $\cdot$) tvoří cyklickou grupu. Z definice tělesa víme, že (R\{0},$\cdot$) tvoří grupu. Vyberme z množiny R\{0} prvek α, nejvyššího řádu r, kde $1 \leq r \leq q-1$. Množina R\{0} obsahuje $q-1$ různých prvků. Dá se ukázat ([3]), že řád každého z těchto $q-1$ prvků dělí r, jinak bychom dostali spor s předpokladem, že r je největší. To ale znamená, že každý z $q-1$ různých prvků β je kořenem rovnice $x^r - 1 = 0$, a tedy $x^r - 1$ je dělitelné $\prod_{\beta \in R\setminus\{0\}}(x-\beta)$. Odtud $r \geq q-1$. Jenže $r \leq q-1$, tudíž $r = q-1$. Ukázali jsme, že největší řád prvku je roven velikosti grupy. Tj. grupa je cyklická, tvořená prvky $\alpha, \alpha^2, ..., \alpha^{q-2}, \alpha^{q-1} = 1$ a prvek α, který je generátorem grupy, je současně prvkem primitivním. □

Příklad 2.2.10 Dalším příkladem konečného tělesa, jež v této práci využijeme později, je GF(2^3). Těleso je tvořeno polynomy nad $\mathbb{Z}_2$ modulo ireducibilní polynom třetího stupně, konkrétně $x^3 + x + 1$. Obsahuje prvky $\{0, 1, x, x+1, x^2, x^2+x, x^2+1, x^2+x+1\}$. Charakteristika

tohoto tělesa je $p = 2$. Všechny prvky kromě 0 a 1 mají řád $n = q - 1 = 8 - 1 = 7$, a tudíž jsou všechny primitivní. Uveďme také tabulky sčítání a násobení:

+	0	1	x	x^2	$x+1$	x^2+x	x^2+x+1	x^2+1
0	0	1	x	x^2	$x+1$	x^2+x	x^2+x+1	x^2+1
1	1	0	$x+1$	x^2+1	x	x^2+x+1	x^2+x	x^2
x	x	$x+1$	0	x^2+x	1	x^2	x^2+1	x^2+x+1
x^2	x^2	x^2+1	x^2+x	0	x^2+x+1	x	$x+1$	1
$x+1$	$x+1$	x	1	x^2+x+1	0	x^2+1	x^2	x^2+x
x^2+x	x^2+x	x^2+x+1	x^2	x	x^2+1	0	1	$x+1$
x^2+x+1	x^2+x+1	x^2+x	x^2+1	$x+1$	x^2	1	0	x
x^2+1	x^2+1	x^2	x^2+x+1	1	x^2+x	$x+1$	x	0

Tabulka 5: Sčítání v GF(2^3)

·	0	1	x	x^2	$x+1$	x^2+x	x^2+x+1	x^2+1
0	0	0	0	0	0	0	0	0
1	0	1	x	x^2	$x+1$	x^2+x	x^2+x+1	x^2+1
x	0	x	x^2	$x+1$	x^2+x	x^2+x+1	x^2+1	1
x^2	0	x^2	$x+1$	x^2+x	x^2+x+1	x^2+1	1	x
$x+1$	0	$x+1$	x^2+x	x^2+x+1	x^2+1	1	x	x^2
x^2+x	0	x^2+x	x^2+x+1	x^2+1	1	x	x^2	$x+1$
x^2+x+1	0	x^2+x+1	x^2+1	1	x	x^2	$x+1$	x^2+x
x^2+1	0	x^2+1	1	x	x^2	$x+1$	x^2+x	x^2+x+1

Tabulka 6: Násobení v GF(2^3)

Při práci s Reed-Solomonovými kódy v pozdějších kapitolách pro nás bude výhodné reprezentovat nenulové prvky konečného tělesa jako mocniny primitivního prvku (jak už bylo zmíněno dříve, umocňováním primitivního prvku jsme schopni získat všechny nenulové prvky GF(q)). Vyberme si jeden z primitivních prvků v tělese GF(2^3) (například x) a označme jej α. S využitím tabulky násobení je poměrně snadné určit všechny mocniny tohoto prvku α. Prvkem α^2 rozumíme součin $\alpha \cdot \alpha = x \cdot x = x^2$. Pokračujeme dále s $\alpha^3 = \alpha^2 \cdot \alpha = x^2 \cdot x = x + 1$. Zbylé přiřazení uveďme v tabulce:

α	x	α^3	$x+1$	α^5	x^2+x+1
α^2	x^2	α^4	x^2+x	α^6	x^2+1

Tabulka 7: Přiřazení koeficientu α v GF(2^3)

Nyní, když máme jednotlivé prvky reprezentovány jako mocniny primitivního prvku α, můžeme nahradit také tabulky sčítání a násobení:

$+$	0	1	α	α^2	α^3	α^4	α^5	α^6
0	0	1	α	α^2	α^3	α^4	α^5	α^6
1	1	0	α^3	α^6	α	α^5	α^4	α^2
α	α	α^3	0	α^4	1	α^2	α^6	α^5
α^2	α^2	α^6	α^4	0	α^5	α	α^3	1
α^3	α^3	α	1	α^5	0	α^6	α^2	α^4
α^4	α^4	α^5	α^2	α	α^6	0	1	α^3
α^5	α^5	α^4	α^6	α^3	α^2	1	0	α
α^6	α^6	α^2	α^5	1	α^4	α^3	α	0

Tabulka 8: Sčítání prvků v GF(2^3) jako mocnin primitivního prvku α

$\cdot$	0	1	α	α^2	α^3	α^4	α^5	α^6
0	0	0	0	0	0	0	0	0
1	0	1	α	α^2	α^3	α^4	α^5	α^6
α	0	α	α^2	α^3	α^4	α^5	α^6	1
α^2	0	α^2	α^3	α^4	α^5	α^6	1	α
α^3	0	α^3	α^4	α^5	α^6	1	α	α^2
α^4	0	α^4	α^5	α^6	1	α	α^2	α^3
α^5	0	α^5	α^6	1	α	α^2	α^3	α^4
α^6	0	α^6	1	α	α^2	α^3	α^4	α^5

Tabulka 9: Násobení prvků v GF(2^3) jako mocnin primitivního prvku α

Kdykoliv jsme doteď konstruovali konečné těleso, vždy jsme si dopředu uvedli ireducibilní polynom, vzhledem ke kterému jsme polynomy sčítali nebo násobili modulo. Vyvstává otázka, jak takový polynom nalézt, když jej dopředu neznáme.

Jako jednoduchý se může jevit způsob, kdy nejprve z velikosti konečného tělesa zjistíme alespoň stupeň, jaký bude ireducibilní polynom mít. Poté bychom vypsali všechny možné polynomy tohoto stupně a následně ověřili, zda jsou či nejsou ireducibilní. Metody ověřování ovšem sahají nad rámec této práce, a tak by nám nezbývalo nic jiného, než hrubou silou zkoušet všechny potenciální dělitele.

Navíc čím větší stupeň máme, tím více možných polynomů musíme ověřit (pro polynomy pátého stupně s koeficienty z GF(2) existuje již 2^5 různých polynomů). Zkoumat tolik polynomů ručně s využitím tužky a papíru je zjevně značně nevýhodné. Existuje ovšem spousta matematických softwarů (například MATLAB), jež mají integrovány funkce, které jsou schopny pro požadované konečné těleso vrátit ireducibilní polynom. Čtenáře můžeme odkázat na online dokumentaci MATLABu [4], kde se kromě zmíněné funkce nachází také tabulka ireducibilních polynomů pro konečná tělesa GF(2^m), kde $1 \leq m \leq 16$.

3 Samoopravné kódy

V této kapitole vysvětlíme pojem "samoopravný kód" jako takový a zároveň uvedeme základní typy samoopravných kódu tak, abychom mohli korektně zavést a zařadit i Reed-Solomonovy kódy. Hlavním zdrojem informací pro náš text jsou tituly [2] a [5].

3.1 Samoopravný kód

Mějme dva objekty. Jedním z nich bude vysílač a druhým přijímač. Tyto dva objekty mezi sebou komunikují skrze prostředek, který nazveme komunikační kanál. Pokud vše proběhne v pořádku, vysílač bez problémů skrze kanál vyšle informaci přijímači, který ji přijme a zpracuje. Co když však kdekoliv po cestě dojde k chybě a informace nedorazí v takovém stavu, v jakém vysílač zamýšlel?
Pokud by se jednalo o komunikaci dvou lidí, člověk, který poslouchal (přijímač), si jednoduše vyžádá, aby byla zpráva zopakována. Ne vždy však takovou možnost máme, u přenosu informací z kosmu si je jednoduše nemůžeme dovolit posílat na vícekrát.
V takovém případě máme dvě možnosti. První z nich je zajistit bezchybný přenos. Tato možnost je však z pochopitelných důvodů poměrně těžce realizovatelná. Ta druhá je pak zajistit, že i poté, co nastane chyba, bude přijímač schopen chybu odhalit a zprávu vrátit do původní podoby. Prostředkem k takovému ošetření jsou samoopravné kódy.

Příklad 3.1.1 Uvažujme takový kanál, po kterém přenášíme pouze čísla 0 nebo 1. Přijímač má nulovou šanci odhalit, zda opravdu dostal odeslanou cifru. Využijeme proto tzv. **binární opakovací kód**. Číslo jedna se rozhodneme "prodloužit" a reprezentovat jej jako 111. Číslo nula pak jako 000. Pokud odešleme 000 a při přenosu nastane jedna chyba, přijímač přijme například zprávu 001. Přijímač ale ví, že vysílač mohl odeslat pouze 000 nebo 111, a proto odhalí, že došlo k chybě a zprávu správně opraví na 000. Pokud ovšem nastanou dvě chyby, bohužel bude zpráva dekódována chybně.

Na příkladu je vidět, jak jsme pomocí vhodně vymyšleného prodloužení (tzv. **redundance**) zajistili vyšší pravděpodobnost úspěšného přijetí.

Nyní je třeba formulovat jakési vlastnosti, kterými jsme schopni samoopravné kódy popisovat a odlišovat.

Jistě nás například zajímá, jak dlouhou zprávu jsme najednou schopni odeslat. Délku zprávy označme jako k. Dále jsme si ovšem řekli, že abychom byli schopni opravovat chyby, zprávu je nutné prodloužit o redundanci. Proto po přičtení redundance dostáváme celkovou délku kódového slova, kterou označme n. Od toho, jak moc jsme zprávu prodloužili, se pak odvíjí parametr t, určující, kolik chyb může nastat, abychom byli pořád schopni zprávu korektně dekódovat.

Když zmíněné parametry vztáhneme na náš příklad, délka původní zprávy k bude rovna 1, protože jsme chtěli odesílat pouze 0 nebo 1. Pak byla zpráva prodloužena o redundanci na celkovou délku n rovnu 3. Tím jsme dostali schopnost opravit jednu nastalou chybu. Parametr t je tedy roven 1.

Další důležitou veličinou, kterou je nutné umět popsat, je samotná chyba, která při přenosu nastala. K popisu využijeme následující definice.

Zároveň je dobré zmínit, že je pro nás žádoucí kódová slova chápat jako vektory. Získáme tím vlastnosti, které nám umožní s nimi lépe pracovat.

Definice 3.1.2 *Hammingova vzdálenost d dvou vektorů $\vec{x}$ a $\vec{y}$ je rovna počtu souřadnic, ve kterých se liší.*

Definice 3.1.3 *Hammingova váha w vektoru $\vec{x}$ je rovna počtu nenulových souřadnic.*

Když se opět vrátíme k příkladu **3.1.1**, odeslané kódové slovo 000 si můžeme označit jako vektor $\vec{x}$ a přijaté chybné slovo 001 jako vektor $\vec{y}$. Hammingova vzdálenost mezi nimi je $d(\vec{x}, \vec{y}) = 1$, protože se liší v jedné souřadnici. Hammingova váha vektoru $\vec{y}$ je pak $w(\vec{y}) = 1$, protože obsahuje jednu nenulovou souřadnici.

Hammingovu vzdálenost využil v našem příkladě i přijímač, který po zjištění, že nedostal jedno ze dvou kódových slov, dekódoval zprávu tak, aby vzdálenost mezi kódovým slovem a přijatým slovem byla co nejmenší (jedná se o tvz. **"nearest neighbour decoding"**, neboli dekódování k nejbližšímu sousedovi). Proto by také dekódoval špatně, kdyby během přenosu nastaly dvě chyby. Jednoduše řečeno by chyba byla tak markantní, že by přijaté slovo mělo "blíže" k jinému slovu, než bylo odesláno. Z toho nám jasně vyplývá, že čím spolehlivější dekódování požadujeme, tím vzálenější od sebe kódová slova musí být.

Nyní se budeme chvíli zabývat samotnou schopností opravovat chyby (parametr t). Vyvstává totiž otázka, jak poznat, kolik chyb si pro daný kód můžeme dovolit a také jak volit redundanci tak, abychom měli jistotu, že daný kód bude opravovat tolik chyb, kolik bychom si přáli. Než se však do toho pustíme je zároveň nutné si uvědomit, že ne vždy potřebujeme chyby umět opravit. V případech, kdy máme možnost zažádat o přeposlání zprávy, nám stačí, když jsme schopni pouze chybu odhalit. Je tedy nutné rozlišovat schopnost *detekovat* chyby a *opravovat* chyby. Rozdíl si ukážeme na následujících dvou příkladech.

Příklad 3.1.4 Uvažujme následující kód C, tvořen kódovými slovy 000, 011, 101 a 110. Čtenář si může sám ověřit, že ať si vybereme jakékoliv kódové slovo a v rámci simulace chyby v něm

změníme libovolnou jednu cifru, nikdy nedostaneme jiné kódové slovo. Nastane-li jedna chyba, přijímač je schopen ji zaregistrovat a vyžádat si, aby mu byla zpráva poslána znovu. Zároveň si můžeme všimnout, že jelikož už teď víme, že přijímač dekóduje chybné slovo na nejbližší možné kódové slovo, dostane se do patové situace, jelikož existuje více možností takovéhoto dekódování. O daném kódu můžeme prohlásit, že je schopen detekovat jednu nastalou chybu a zároveň není schopen žádnou opravit.

Příklad 3.1.5 Kód z předchozího příkladu vhodně prodloužíme o dvě cifry tak, že kódová slova nyní budou 00000, 01101, 10110 a 11011. Čtenář si opět může vyzkoušet, že tento kód už je nyní schopen i jednu chybu opravit. Detekovat je dokonce schopen chyby dvě.

Jak už bylo naznačeno dříve, je žádoucí umět manipulovat s konstrukcí kódu tak, aby například opravil předem určený počet chyb. Následující vztahy a definice nám s tím mohou pomoci.

Definice 3.1.6 *Minimální vzdálenost kódu C, značená d(C), je* $\min\{d(\vec{x},\vec{y})|\vec{x},\vec{y}\in C,\vec{x}\neq\vec{y}\}$.

Pro demonstraci můžeme využít kód z příkladu **3.1.4**, kde vidíme, že nejmenší možný počet souřadnic, ve kterých se dvě kódová slova liší, je 2, proto $d(C) = 2$. Pro kód z příkladu **3.1.5** je $d(C) = 3$.

Věta 3.1.7 [5] *Kód C je schopen detekovat s chyb, pokud platí* $d(C) \geq s + 1$.

Důkaz Předpokládejme, že $d(C) \geq s + 1$ platí. Dále předpokládejme, že bylo odesláno kódové slovo $\vec{x}$ a nastalo s nebo méně chyb. Pak jistě slovo, které jsme přijali, nemůže být jiné kódové slovo, a tak můžeme detekovat chybu. □

Věta 3.1.8 [5] *Kód C je schopen opravit t chyb, pokud platí* $d(C) \geq 2t + 1$.

Důkaz Předpokládejme, že $d(C) \geq 2t + 1$ platí. Dále předpokládejme, že bylo odesláno kódové slovo $\vec{x}$, přijato slovo $\vec{y}$ a nastalo t nebo méně chyb tak, že $d(\vec{x},\vec{y}) \leq t$. Pokud je vektor $\vec{z}$ nějaké jiné kódové slovo než $\vec{x}$, pak platí $d(\vec{z},\vec{y}) \geq t + 1$, protože jinak by muselo platit $d(\vec{z},\vec{y}) \leq t$, což by podle trojúhelníkové nerovnosti znamenalo, že $d(\vec{x},\vec{z}) \leq d(\vec{x},\vec{y}) + d(\vec{z},\vec{y}) \leq 2t$, což je v rozporu s $d(C) \geq 2t+1$. Z toho plyne, že $\vec{x}$ je nejbližší kódové slovo k $\vec{y}$ a tudíž jsme opravili chyby. □

Definice 3.1.9 *Kód C s délkou kódového slova n, délkou zprávy k a minimální vzdáleností d se nazývá (n,k,d)-kód.*

Pro ucelenost nyní na příkladech **3.1.4** a **3.1.5** shrňme nejdůležitější vlastnosti (n, k, d)-kódů. Pokud je minimální vzdálenost d sudá, je obecný (n, k, d)-kód schopen opravit až $t = (d - 2)/2$

chyb. Pokud lichá, pak kód opraví až $t = (d-1)/2$ chyb. Pro oba případy můžeme s využitím tzv. dolní celé části (zapisujeme $\lfloor x \rfloor$) jednotně zapsat, že (n, k, d)-kód opraví $t = \lfloor (d-1)/2 \rfloor$ chyb. Detekovat je v obou případech schopen $s = d - 1$ chyb.

Kód $C = \{000, 011, 101, 110\}$ z příkladu **3.1.4** je (3,2,2)-kód. Je schopen detekovat $s = d - 1 = 2 - 1 = 1$ chybu a není schopen opravit žádnou chybu ($t = \lfloor (d-1)/2 \rfloor = \lfloor (2-1)/2 \rfloor = 0$).

Kód $C = \{00000, 01101, 10110, 11011\}$ z příkladu **3.1.5** je (5,2,3)-kód. Je schopen detekovat $s = d - 1 = 3 - 1 = 2$ chyby a opravit $t = \lfloor (d-1)/2 \rfloor = \lfloor (3-1)/2 \rfloor = 1$ chybu.

3.2 Hlavní problém teorie kódování

Důležitým problémem teorie kódování je určování kvality kódu. Bude nás zajímat, proč je daný kód dobrý a jak poznáme, že je některý kód lepší než jiný.

Dobrý kód umožňuje rychlý přenos informací, což ve výsledku znamená, že není přílíš dlouhý (velikost zprávy k se příliš neliší od celkové délky n). Zároveň je žádoucí, aby byl spolehlivý, tudíž aby opravil co nejvíce chyb (měl co největší d). V neposlední řadě je důležité, aby měl dobrý kód co nejvíce kódových slov (počet kódových slov kódu C označme jako $|C|$), jelikož chceme mít možnost odeslat co nejvíce různých zpráv. Pokud tedy o nějakém kódu prohlásíme, že je "silnější" než jiný, myslíme tím, že má lepší právě tyto vlastnosti.

Bohužel jsou tyto tři požadavky vzájemně protichůdné. Když totiž požadujeme větší spolehlivost, potřebujeme zvětšit minimální vzdálenost, čehož můžeme docílit dvěma způsoby. Buď zvětšíme redundanci, nebo zmenšíme počet kódových slov. Zvětšením redundance však prodloužíme přenos a menší počet kódových slov snižuje variabilitu zpráv. Naopak snížením redundance snižujeme spolehlivost a tak dále. V praxi vše funguje tak, že se pro dva libovolné fixně dané parametry snažíme najít kód s nejlepším možným třetí parametrem. Tomuto problému se obecně říká **hlavní problém teorie kódování** (z angl. *main coding theory problem*). Nejčastější variantou je nalézt kód s největším počtem kódových slov $|C|$ pro pevně danou délku n a minimální vzdálenost d.

Tento maximální počet kódových slov, pro který existuje kód s parametry n a d, definovaný nad abecedou o velikosti q, označme $A_q(n, d)$. Pro řadu hodnot n a d je hodnota $A_q(n, d)$ známa přesně (viz [5]). V případech, kdy hodnota určena přesně není, ji můžeme alespoň odhadnout. K tomu využijeme tzv. **Singletonův odhad**, pojmenovaný podle *R.C.Singletona*, který svůj výsledek publikoval v [6].

Věta 3.2.1 [7] *Mějme (n,k,d)-kód C, definovaný nad abecedou symbolů o velikosti q. Pak platí, že* $A_q(n, d) \leq q^{n-d+1}$.

Důkaz Pokud máme slova délky n, definovaná nad abecedou o velikosti q, celkový počet možností, jakých různých slov můžeme dosáhnout je q^n. Nyní vezměme libovolný (n,k,d)-kód C nad abecedou o velikosti q. Víme jistě, že všechna kódová slova jsou různá. Pokud všechna slova zkrátíme o prvních $d-1$ symbolů, všechna tato nově vzniklá kódová slova musí být pořád různá, protože minimální vzdálenost původního kódu je d. Počet nových kódových slov se oproti původnímu kódu nemění. Nově vzniklá slova mají zjevně délku $n-(d-1) = n-d+1$. A tudíž jich může být nanejvýš q^{n-d+1}. A protože byl kód C zvolen libovolně, musí tento odhad platit i pro největší možný kód o těchto parametrech, a tudíž $|C| \leq A_q(n,d) \leq q^{n-d+1}$. □

Příklad 3.2.2 Pokud si stanovíme, že chceme například sestrojit binární ($q = 2$) kód, který bude mít délku kódového slova $n = 10$ a minimální vzdálenost $d = 5$, podle Singletonova odhadu s jistotou víme, že nenajdeme kód, který by měl více než $2^{10-5+1} = 64$ kódových slov. Jedná se však pouze o odhad a není v žádném případě zaručeno, že binární kód s našimi stanovenými parametry, který má 64 kódových slov opravdu existuje.

Kódy, jež v Singletonově odhadu splňují rovnost (tzn. počet slov $|C| = q^{n-d+1}$), se nazývají **MDS-kódy** (*z angl. Maximum Distance Separable*). Reed-Solomonovy kódy, které jsou hlavním předmětem naší práce, do této třídy kódů spadají.

Dalším kritériem, podle kterého se dají různé třídy kódů srovnávat, je časová složitost, případně paměťová náročnost jejich kódovacích a dekódovacích algoritmů. V následujících kapitolách například jasně uvidíme, že přechod na užší třídu kódů nám výrazně zredukuje množství dat, které je při dekódování nutno uchovávat v paměti.

Uvedená kritéria jsou hlavní motivací, proč přecházet na třídy kódů (v našem případě Reed-Solomonovy kódy), za nimiž často stojí složitější matematický aparát. Získáme tím totiž možnost flexibilnější volby parametrů, jelikož jejich kódovací a dekódovací techniky umožňují snazší a paměťově méně náročnější práci s delšími a objemnějšími kódy a také schopnost opravovat více chyb.

3.3 Lineární kód

Jedním ze základních druhů kódů jsou **lineární kódy**. Ke kódování a dekódování využívají aparát lineární algebry. Předpokládá se znalost vektorů, vektorových prostorů a podprostorů.

Definice 3.3.1 *Uvažujme vektorový prostor $V(n,q)$ sestrojený nad konečným tělesem $GF(q)$, kde n je přirozené číslo určující dimenzi prostoru a q prvočíslo či mocnina prvočísla představující počet symbolů abecedy. Kód C jehož množina slov je podprostorem v prostoru $V(n,q)$ se nazývá lineární kód.*

Z definice plyne, že kód C je lineární tehdy a jen tehdy, platí-li:

- Součtem libovolných kódových slov z C dostáváme vždy kódové slovo.
- Vynásobením libovolného prvku z GF(q) a kódového slova z C dostáváme opět kódové slovo.
- Kód C obsahuje nulový vektor jako jedno z kódových slov.

Příklad 3.3.2 Zaveďme si kód C, jehož kódová slova budou 00000, 01101, 10110 a 11011. Vezmeme-li si například slova 01101 a 10110, jejich součtem dostaneme 11011, což je kódové slovo. Jednoduše si lze ověřit, že vše platí i pro jiná dvě libovolná slova z C. Zároveň pokud jakékoliv kódové slovo přenásobíme 0 nebo 1, dostáváme taktéž kódové slovo. Kód C je proto zjevně lineární. Kód budeme nyní využívat ve všech dalších příkladech této kapitoly.

Pokud je kód C podprostor dimenze k vektorového prostoru $V(n, q)$, pak kód označujeme jako $[n, k]$-kód. Pokud navíc chceme uvést minimální vzdálenost, můžeme kód označit jako $[n, k, d]$-kód.

Poznámka V tomto textu budeme značení lineárního kódu odlišovat od obecného značení samoopravného kódu použitím "hranatých" závorek.

Lineární kódy mají jednu velice důležitou vlastnost, která je obsahem následující věty.

Věta 3.3.3 [5] *Minimální vzdálenost d(C) lineárního kódu je rovna nejmenší váze w(C) nenulových kódových slov kódu C.*

Důkaz Mějme dva kódové vektory $\vec{x}$ a $\vec{y}$ z C, které jsou vůči sobě v nejmenší vzdálenosti. Pak platí, že $d(\vec{x}, \vec{y}) = w(\vec{x} - \vec{y})$, protože vektor $\vec{x} - \vec{y}$ má nenulové souřadnice právě v těch místech, ve kterých se vektory $\vec{x}$ a $\vec{y}$ liší. Jelikož je vektor $\vec{x} - \vec{y}$ kódovým slovem, dostáváme vztah

$$d(C) = w(\vec{x} - \vec{y}) \geq w(C).$$

Naopak pro nějaký kódový vektor $\vec{x} \in C$ platí, že

$$w(C) = w(\vec{x}) = d(\vec{x}, \vec{0}) \geq d(C),$$

protože $\vec{0}$ určitě patří do C.

Tudíž $d(C) \geq w(C)$ a $w(C) \geq d(C)$, z čehož vyplývá, že $d(C) = w(C)$. □

Příklad 3.3.4 Pokud se vrátíme k našemu kódu z příkladu **3.3.2**, vidíme, že slova s nejmenší váhou (tzn. s nejmenším počtem jedniček) jsou 01101 a 10110. Jejich váha je rovna 3. Pokud spolu porovnáme všechna kódová slova, zjistíme, že nejmenší nalezená vzdálenost (tzn. nejmenší počet souřadnic, ve kterých se liší) je také rovna třem.

Vyvstává otázka, proč bychom měli (a proč neměli) linární kódy používat. Pokusíme se přiblížit výhody a nevýhody, které přináší.

První z výhod byla již uvedena ve větě **3.3.3**. Za normálních okolností, pokud chceme nalézt minimální vzdálenost nějakého nelineárního kódu, nám nezbývá nic jiného, než porovnat každé nenulové kódové slovo s každým. U lineárních kódů nám však stačí prozkoumat pouze váhy nenulových slov, čímž si poměrně podstatně zmenšíme počet provedených operací.

Pro plnou specifikaci nelineárního kódu je nutné uvádět veškerá kódová slova, která obsahuje. Druhou výhodou lineárních kódů je, že všechna slova uvádět nemusíme. Plně nám postačí báze.

Definice 3.3.5 *Matice G o velikosti $k \times n$, jejíž řádky tvoří bázi lineárního $[n, k]$-kódu, se nazývá generující matice.*

Příklad 3.3.6 Náš lineární [5,3]-kód z příkladu **3.3.2** má generující matici

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Slovo 00000 není třeba uvádět, protože jistě víme, že se v kódu vyskytuje. Slovo 11011 pak nemusíme uvádět, protože jej odvodíme součtem dvou řádků z matice G.

Třetí výhodou lineárních kódů je pak poměrně jednoduché kódování a dekódování, kterému se budeme věnovat v kapitole **3.4**.

Nevýhodami může být, že neexistuje lienární kód definovaný nad abecedou symbolů, jejíž velikost není prvočíslo nebo mocnina prvočísla a také, že lineární kódy často nemusí být dostatečně "silné", jelikož dekódovací algoritmy jsou pro příliš velké kódy znatelně neefektivní.

3.4 Kódování a dekódování lineárních kódů

Kódování zprávy do kódového slova lineárního kódu není vůbec složité.

Mějme lineární [n,k]-kód C definovaný nad GF(q) s generující maticí G. C obsahuje q^k kódových slov, a tudíž jsme schopni odeslat q^k různých zpráv. Tyto zprávy budou všechny *k-tice*

(budou mít k složek) vektorového prostoru V(k,q). Konkrétní zprávu $\vec{m} = (m_1, m_2, ..., m_k)$ zakódujeme prostým vynásobení zprava generující maticí G. Výsledkem pak bude kódové slovo z C.

Lineární $[n, k]$-kód C nad GF(q) proto můžeme alternativně definovat jako množinu q^k kódových slov ve formě řádkového vektoru $\vec{c} = (c_0, c_1, ..., c_{n-1})$, kde $c_i \in GF(q)$, přičemž kódová slova jsou definována lineární transformací $\vec{c} = \vec{m} \cdot G$, kde G je generující matice řádu $k \times n$ o hodnosti k.

Příklad 3.4.1 Kód z příkladu **3.3.2** je lineární [5,2]-kód definovaný nad GF(2). Obsahuje 2^2 kódových slov, je tedy schopný odeslat 4 různé zprávy, kterými jsou dvousložkové vektory prostoru V(2,2). Konkrétně se jedná o zprávy 00, 01, 10 a 11. Vybereme si například zprávu 01 a zakódujeme ji:

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Tedy kódové slovo $\vec{c}$, odpovídající zprávě $\vec{m} = [01]$, je $\vec{c} = [10110]$.

Dekódování je už o něco složitější, nicméně nijak zásadně.

Předpokládejme, že bylo odesláno kódové slovo $\vec{c}$, po cestě nastala chyba a přijato bylo slovo $\vec{r}$. **Chybový vektor** $\vec{e}$ definujeme jako rozdíl přijatého slova a původního kódového slova. Úkolem dekodéru je pak odhalit vektor $\vec{c}$ na základě přijatého vektoru $\vec{r}$, případně vypátrat chybový vektor $\vec{e}$. Řešení nabízí využití tzv. **"cosetů"**[1].

Definice 3.4.2 *Mějme [n,k]-kód C definovaný nad GF(q) a libovolný vektor $\vec{a}$ z V(n,q). Pak množina $\vec{a} + C = \{\vec{a} + \vec{x} \mid \vec{x} \in C\}$ se nazývá coset C.*

Lemma 3.4.3 [5] *Předpokládejme, že $\vec{a} + C$ je coset a že $\vec{b} \in \vec{a} + C$, pak $\vec{a} + C = \vec{b} + C$.*

Důkaz Protože $\vec{b} \in \vec{a} + C$, dostáváme $\vec{b} = \vec{a} + \vec{x}$, pro nějaké konkrétní $\vec{x} \in C$. Libovolný vektor z $\vec{b} + C$ lze zapsat jako $\vec{b} + \vec{y},\ \forall \vec{y} \in C$. Platí

$$\vec{b} + \vec{y} = (\vec{a} + \vec{x}) + \vec{y} = \vec{a} + (\vec{x} + \vec{y}) \in \vec{a} + C.$$

Tudíž $\vec{b} + C \subseteq \vec{a} + C$. Naopak pokud $\vec{a} + \vec{z} \in \vec{a} + C$, potom

$$\vec{a} + \vec{z} = (\vec{b} - \vec{x}) + \vec{z} = \vec{b} + (\vec{z} - \vec{x}) \in \vec{b} + C.$$

Tudíž $\vec{a} + C \subseteq \vec{b} + C$, a proto $\vec{b} + C = \vec{a} + C$. □

[1] Slovo coset je převzato z angličtiny. Českým ekvivalentem je označení "třída rozkladu grupy". V této práci však zůstaneme u označení coset.

Další užitečné vlastnosti cosetů jsou obsahem tzv. **Lagrangeovy věty** (viz [1]).

Věta 3.4.4 [5] *Mějme [n,k]-kód C definovaný nad GF(q). Pak*

1. *Každý vektor z V(n,q) patří do nějakého cosetu C*
2. *Každý coset obsahuje přesne q^k vektorů*
3. *Dva cosety jsou buď zcela stejné, nebo zcela různé (nenastane situace, kdy by sdílely pouze některé prvky)*

Důkaz

1. Pokud $\vec{a} \in V(n,q)$, pak $\vec{a} = \vec{a} + \vec{0} \in \vec{a} + C$
2. Zobrazení z C do $\vec{a}+C$, definované jako $\vec{x} \to \vec{a}+\vec{x}$ pro všechna $\vec{x} \in C$, je zjevně injektivní. Tudíž $|\vec{a} + C| = |C| = q^k$
3. Předpokládejme, že se cosety $\vec{a} + C$ a $\vec{b} + C$ částečně překrývají. Pak pro nějaký vektor $\vec{v}$ dostáváme $\vec{v} \in (\vec{a}+C) \cap (\vec{b}+C)$. Tím pádem pro nějaké dva vektory $\vec{x}$ a $\vec{y} \in C$ dostáváme $\vec{v} = \vec{a} + \vec{x} = \vec{b} + \vec{y}$. Tudíž $\vec{b} = \vec{a} + (\vec{x} - \vec{y}) \in \vec{a} + C$, což podle Lemmatu 3.4.3 znamená, že $\vec{b} + C = \vec{a} + C$ □

Poznámka Z věty vyplývá, že cosety tvoří "rozklad" na množině všech kódových slov kódu C.

Příklad 3.4.5 Po sjednocení všech cosetů kódu C z příkladu **3.3.2** dostaneme celý kód C, tudíž vektorový prostor V(5,2). Nyní na ukázku vypíšeme alespoň některé cosety :

$$00000 + \mathrm{C} = \{00000, 01101, 10110, 11011\}$$
$$10001 + \mathrm{C} = \{10001, 11100, 00111, 01010\}$$
$$00100 + \mathrm{C} = \{00100, 01001, 10010, 11111\}$$

Definice 3.4.6 *Vektor s nejmenší váhou v cosetu se nazývá coset leader*[1]*. Pokud je takových vektorů více, vybereme náhodně jeden z nich.*

Nyní máme veškeré náležitosti k sestrojení tabulky **standard array**[2], pomocí které můžeme dekódovat. Návod k sestrojení je následujicí:

1. Do prvního řádku vypišme všechna kódová slova kódu C. První bude nulový vektor, zbytek v libovolném pořadí.
2. Vybereme libovolný vektor $\vec{a_1}$, který má minimální váhu a nevyskytuje se v prvním řádku. Napíšeme jej pod nulový vektor. Zbytek řádku doplníme jako coset $\vec{a_1} + C$ tak, aby součet $\vec{a_1} + \vec{x}$ ($\vec{x}$ je z prvního řádku), byl vepsán vždy pod vektor $\vec{x}$.

[1] Jako český ekvivalent termínu coset leader se používá "reprezentant třídy". Zůstaneme u anglického označení se skloňováním podle vzoru pán.

[2] Rovněž zůstaneme u zavedeného anglického pojmu standard array.

3. Vybereme libovolný vektor $\vec{a_2}$, který má minimální váhu a nevyskytuje se v prvním ani druhém řádku. Zbytek řádku doplníme jako coset $\vec{a_2} + C$ stejně tak jako v kroku 2.

4. Pokračujeme jako v předešlých krocích, dokud nejsou vypsány všechny cosety a každý vektor z V(n,q) není uveden právě jednou.

Příklad 3.4.7 Standard array pro kód z příkladu **3.3.2** vypadá následovně:

00000	**01101**	**10110**	**11011**
00001	01100	10111	11010
00010	01111	10100	11001
00100	01001	10010	11111
01000	00101	11110	10011
10000	11101	00110	01011
11000	10101	01110	00011
10001	11100	00111	01010

Tabulka 10: Standard array lineárního [5,3]-kódu

Když máme sestrojenou tabulku, můžeme demonstrovat dekódovací proces. Dejme tomu, že přijmeme slovo $\vec{r} = [11110]$. Slovo nalezneme v tabulce. Nachází se v pátém řádku. Chybový vektor $\vec{e}$ se nachází ve stejném řádku úplně vlevo. V našem případě se jedná o $\vec{e} = [01000]$. Kódové slovo, na které dekódujeme, se pak nachází ve stejném sloupci, jako přijaté slovo, úplně nahoře. Dekódujeme na slovo $\vec{c} = \vec{r} - \vec{e} = [10110]$. Pokud nás nezajímá chybový vektor, dekódovací proces zjednodušíme tak, že najdeme přijaté slovo v tabulce a v prvním řádku nalezneme příslušné kódové slovo ze stejného sloupce.

Ačkoliv je tento dekódovací algoritmus poměrně jednoduchý na provedení, není úplně ideální pro velké kódy. Sestavovat a především udržovat v paměti obrovskou kódovací tabulku je neefektivní. Nabízí se možnost využít dalších vlastností lineárních kódů, které tyto problémy do jisté míry řeší. Nejdřív ovšem musíme nadefinovat nové pojmy.

Definice 3.4.8 *Mějme lineární [n,k]-kód C s generující maticí G. Kontrolní maticí H nazveme matici o velikosti* $(n-k) \times n$*, která splňuje rovnost* $GH^T = O$*, kde* H^T *představuje transponovanou matici H a O představuje nulovou matici o velikosti* $k \times (n-k)$*.*

Poznámka Alternativně můžeme kontrolní matici kódu C nadefinovat jako generující matici tzv. **duálního kódu** k C. Duální kód k C je definován jako kód, jehož všechna kódová slova jsou ortogonální se všemi kódovými slovy z C. Pojem je více rozveden například v [5].

Pro jeden z možných způsobů hledání kontrolní matice musíme generující matici nejdříve převést do tzv. **standardního tvaru** $G = [I_k|A]$, kde I_k představuje jednotkovou matici o velikosti $k \times k$ a A libovolnou odpovídající matici o velikosti $k \times (n-k)$.

Převodu docílíme jednoduše s využitím ekvivalentních řádkových úprav (tzv. Gauss-Jordanova metoda viz [8]) s tím rozdílem, že samozřejme používáme modulární aritmetiku. Řádkové úpravy provedené na generující matici nemají vliv na výslednou množinu kódových slov (kód zůstává stejný). Současně je možné měnit pořadí sloupců. V takovém případě dostáváme kód o stejných parametrech, který je izomorfní (ekvivalentní) s původním kódem.

Příklad 3.4.9 Generující matici kódu z příkladu **3.3.2** převedeme pouhým prohozením řádků:

$$G = \left[\begin{array}{cc|ccc} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{array}\right] \sim \left[\begin{array}{cc|ccc} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{array}\right] \sim [I_2|A]$$

Věta 3.4.10 [5] *Mějme generující matici G lineárního [n,k]-kódu C, přičemž matice je ve standardním tvaru* $G = [I_k|A]$. *Pak kontrolní matice H má tvar* $H = [-A^T|I_{n-k}]$.

Důkaz Generující matice G má tvar:

$$G = \left[\begin{array}{ccc|ccc} 1 & & 0 & a_{1,1} & \cdots & a_{1,n-k} \\ & \ddots & & \vdots & & \vdots \\ 0 & & 1 & a_{k,1} & \cdots & a_{k,n-k} \end{array}\right].$$

Předpokládejme, že matice H má tvar:

$$H = \left[\begin{array}{ccc|ccc} -a_{1,1} & \cdots & -a_{k,1} & 1 & & 0 \\ \vdots & & \vdots & & \ddots & \\ -a_{1,n-k} & \cdots & -a_{k,n-k} & 0 & & 1 \end{array}\right].$$

Matice H zjevně dosahuje rozměrů kontrolní matice a její řádky jsou lineárně nezávislé. Součin libovolného i-tého řádku matice G s libovolným j-tým řádkem matice H vypadá následovně:

$$0 + ... + 0 + (-a_{i,j}) + 0 + ... + 0 + a_{i,j} + 0 + ... + 0 = 0.$$

Z toho vyplývá, že každý řádek matice H je ortogonální s každým řádkem matice G, a tudíž matice H je opravdu kontrolní maticí. □

Příklad 3.4.11 Kontrolní matice ke kódu z příkladu **3.3.2** je

$$H = [-A^T|I_3] \sim \left[\begin{array}{cc|ccc} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Poznámka Počítáme s binárním kódem (tzn. kódem definovaným nad abecedou z GF(2)), proto není potřeba řešit známenko mínus.

Definice 3.4.12 *Mějme kontrolní matici H [n,k]-kódu C a libovolný vektor $\vec{r} \in C$. Pak řádkový vektor o velikosti $1 \times (n-k)$ definovaný jako $S(\vec{r}) = \vec{r}H^T$ se nazývá syndrom vektoru $\vec{r}$.*

Důležitou vlasností syndromu je, že pokud je vektor $\vec{r}$ zároveň kódovým slovem, pak jeho syndrom bude roven nulovému vektoru.

Lemma 3.4.13 [5] *Dva vektory $\vec{u}$ a $\vec{v}$ jsou ve stejném cosetu tehdy a jen tehdy, je-li jejich syndrom stejný.*

Důkaz Vektory $\vec{u}$ *a* $\vec{v}$ jsou ve stejném cosetu, a proto

$$\vec{u} + C = \vec{v} + C \Leftrightarrow \vec{u} - \vec{v} \in C \Leftrightarrow (\vec{u} - \vec{v})H^T = 0 \Leftrightarrow \vec{u}H^T = \vec{v}H^T \Leftrightarrow S(\vec{u}) = S(\vec{v}). \qquad \square$$

S pomocí syndromů teď můžeme dříve ukázaný dekódovací algoritmus zjednodušit:

1. Přijmeme slovo $\vec{r}$ a vypočteme jeho syndrom $S(\vec{r}) = \vec{r}H^T$.
2. Syndrom nalezneme v tabulce a tím získáme odpovídajícího coset leadera, a tudíž i chybový vektor $\vec{e}$.
3. Dekódujeme na kódové slovo $\vec{c} = \vec{r} - \vec{e}$.

Příklad 3.4.14 Nejprve spočteme syndromy všech coset leaderů kódu C (levý sloupec tabulky z příkladu **3.4.7**)

S(00000) = 000	S(01000) = 101
S(00001) = 001	S(10000) = 110
S(00010) = 010	S(11000) = 011
S(00100) = 100	S(10001) = 111

Nyní můžeme původní tabulku zkrátit tak, že uvedeme pouze prvky levého sloupce a jejich syndromy.

Coset leadeři	Syndromy
00000	000
00001	001
00010	010
00100	100
01000	101
10000	110
11000	011
10001	111

Tabulka 11: Syndromová tabulka lineárního [5,3]-kódu

Po přijetí nějakého slova vypočteme jeho syndrom, následně nalezneme příslušný řádek a pak od přijatého slova odečtme odpovídajícího coset leadera. Pokud přijmeme slovo $\vec{r} = [11110]$, vypočteme jeho syndrom $S(111110) = \vec{r}H^T = [101]$. Jedná se o pátý řádek tabulky a provedeme proto výpočet $\vec{c} = \vec{r} - \vec{e} = [11110] - [01000] = [10110]$. Dekódujeme na slovo $\vec{c} = [10110]$. Ušetříme paměť, jelikož nám stačí uchovávat mnohem menší tabulku než v případě dekódování pomocí standard array.

Pro mnohem znatelnější demonstraci rozdílů mezi oběma metodami využijme lineární (32,6)-kód, definovaný nad GF(2). Při dekódování pomocí standard array bychom v paměti potřebovali uchovat 2^{n-k} cosetů a 2^k slov o n bitech. Po dosazení konkrétních hodnot zjišťujeme, že náš požadavek na paměť představuje zhruba $2^{32} \cdot 32 = 2^{34} \cdot 8 \doteq 16$ Gigabytů.

Naopak při dekódovaní pomocí syndromů potřebujeme uchovat 2^{n-k} coset leaderů a 2^{n-k} syndromů, tudíž $2^{n-k} \cdot n + 2^{n-k} \cdot (n-k)$ bitů. Konkrétně požadujeme $2^{26} \cdot 32 + 2^{26} \cdot 26 \doteq 500$ Megabytů paměti, tudíž zhruba 32-krát méně než u standard array dekódování.

3.5 Cyklický kód

V minulé kapitole jsme si ukázali dekódování lineárních kódů pomocí tabulky, které bylo pro velké kódy paměťově značně nevýhodné. Tabulku se podařilo podstatně redukovat s pomocí syndromů, ovšem stále se jedná o poměrně velkou tabulku. Je pochopitelné, že vyvstala otázka, zda není možné paměť ušetřit ještě více. S odpovědí přišel *E. Prange*, když v roce 1957 objevil další důležitou třídu kódů. Svůj objev publikoval v [9]. Jedná se o podtřídu lineárních kódů, tzv. **cyklické kódy**. Protože třída cyklických kódů obsahuje další zajímavé podtřídy, mimo jiné také RS-kódy, zavedeme základní pojmy týkající se těchto kódů a ukážeme metody kódování a dekódování. Při formulacích definic a vět této podkapitoly vycházíme především z [2].

Než budeme moci definovat samotný cyklický kód, hodí se nám nejdříve zavést tzv. **cyklický posun** (z anglického "cyclic shift").

Definice 3.5.1 ***Cyklickým posunem*** *souřadnic vektoru $\vec{v}$ o jednu souřadnici doprava rozumíme operaci definovanou takto:*

$$\vec{v} = (v_0, v_1, ..., v_{n-2}, v_{n-1}) \longrightarrow \vec{v}^{(1)} = (v_{n-1}, v_0, v_1, ..., v_{n-2}).$$

Příklad 3.5.2 Pokud si například zvolíme vektor $\vec{v} = [011001]$, jeho cyklický posun bude $\vec{v}^{(1)} = [101100]$.

Nyní můžeme definovat, co je to cyklický kód.

Definice 3.5.3 *Kód C nazveme cyklikcký, pokud:*

- *Je lineární.*
- *Cyklické posuny všech kódových slov jsou taktéž kódová slova.*

Příklad 3.5.4 Kód C, tvořený kódovými slovy {000,101,011,110}, je cyklický.

Příklad 3.5.5 Kód C, z příkladu **3.3.2** tvořený kódovými slovy {00000,01101,10110,11011} cyklický není.

Dále se budeme zabývat vlastnostmi cyklických kódů.

U cyklických kódů je pro reprezentaci kódových slov výhodné využívat polynomy. Můžeme si to dovolit, jelikož prostor polynomů stupně $n-1$ nad GF(q) je izomorfní s vektorovým prostorem $V(n,q)$.

Vektor o n složkách reprezentujeme polynomem stupně nejvýše $n-1$ následovně:

$$\vec{v} = (v_0, v_1, ..., v_{n-1}) \longleftrightarrow v(x) = v_0 + v_1 x + ... + v_{n-1}x^{n-1}.$$

Příklad 3.5.6 Vektor $\vec{v} = [10011]$ můžeme zapsat polynomem:

$$v(x) = 1 + 0x + 0x^2 + 1x^3 + 1x^4 = 1 + x^3 + x^4.$$

Cyklický posun pak u polynomiální reprezentace kódových slov vypadá takto:

$$v(x) = v_0 + v_1 x + ... + v_{n-2}x^{n-2} + v_{n-1}x^{n-1} \longrightarrow v^{(1)}(x) = v_{n-1} + v_0 x + v_1 x^2 + ... + v_{n-2}x^{n-1}.$$

Vztah mezi polynomem $v(x)$ a posunutým polynomem $v^{(1)}(x)$ pak odpovídá násobení polynomem x, kdy výsledkem je zbytek po dělení $(x^n - 1)$:

$$v^{(1)}(x) = x \cdot v(x) \mod (x^n - 1),$$

což je po menších úpravách vidět z následující rovnice:

$$x \cdot v(x) = v_0 x + ... + v_{n-1} x^n,$$

$$x \cdot v(x) = v_{n-1} + v_0 x + ... + v_{n-1} x^n - v_{n-1},$$

$$x \cdot v(x) = v_{n-1} + v_0 x + ... + v_{n-2} x^{n-1} + v_{n-1}(x^n - 1).$$

Pokud chceme provést cyklický posun víckrát najednou, i-tý posun zapíšeme jednoduše jako:

$$v^{(i)}(x) = x^i \cdot v(x) \mod (x^n - 1).$$

Příklad 3.5.7 Vektor $\vec{v} = [10011]$, reprezentovaný polynomem $v(x) = 1 + x^3 + x^4$, posuneme na $v^{(1)}(x) = x \cdot (1 + x^3 + x^4) \mod (x^5 - 1) = x + x^4 + x^5 \mod (x^5 - 1) = 1 + x + x^4$, což skutečně odpovídá vektoru $\vec{v}^{(1)} = [11001]$.

Nyní se budeme zabývat tím, jak cyklické kódy konstruovat. Při konstrukci můžeme využít speciální polynom, jež je vždy prvkem kódu.

Věta 3.5.8 [2] *V cyklickém kódu C existuje právě jeden monický*[1] *polynom* $g(x) = g_0 + g_1 x + ... + g_{r-1} x^{r-1} + x^r$ *nejmenšího stupně.*

Důkaz Předpokládejme, že máme dva nenulové monické polynomy $g(x) = g_0 + g_1 x + ... + g_{r-1} x^{r-1} + x^r$ a $g'(x) = g'_0 + g'_1 x + ... + g'_{r-1} x^{r-1} + x^r$, které jsou různé a oba nejmenšího stupně. Pak $g(x) - g'(x)$ je polynom, který patří do C a má menší stupeň, což vede ke sporu s tím, že $g(x)$ a $g'(x)$ jsou nejmenšího stupně, tudíž $g(x) = g'(x)$. □

Dále si ukážeme, že každé slovo z C lze napsat jako násobek polynomu z předchozí věty.

Věta 3.5.9 [2] *Mějme cyklický kód C. Pak polynom c(x) stupně* $\deg(c(x)) \leq n - 1$ *je kódovým slovem tehdy a jen tehdy, pokud je* $c(x) = m(x) \cdot g(x)$ *, kde g(x) je monický polynom nejmenšího stupně kódu C a m(x) polynom stupně* $\deg(m(x)) < n - r - 1$.

Důkaz Pokud $\deg(c(x)) \leq n - 1$ a $c(x) = m(x) \cdot g(x)$, pak

$$c(x) = (a_0 + a_1 x + ... + a_{n-r-1} x^{n-r-1}) \cdot g(x) = a_0 g(x) + a_1 x g(x) + ... + a_{n-r-1} x^{n-r-1} g(x).$$

Z toho vyplývá, že $c(x)$ je lineární kombinací $g(x), xg(x), ..., x^{n-r-1} g(x)$, a tudíž i kódovým slovem. Naopak pokud je $c(x)$ kódovým slovem, je možné jej zapsat jako $c(x) = m(x) \cdot g(x) + q(x)$

[1] Monický z anglického *"monic"*, koeficient u největší mocniny je roven 1

(násobek dvou polynomů + zbytkový polynom), kde pro stupeň zbytkového polynomu $q(x)$ platí $\deg(q(x)) < \deg(g(x))$. Dále víme, že $q(x)$ musí být kódový polynom, protože $c(x)$ a $m(x) \cdot g(x)$ jsou kódové polynomy. Tudíž $\deg(q(x)) = 0$, protože $\deg(q(x)) < \deg(g(x))$. Dochází ke sporu, jelikož $g(x)$ má nejmenší stupeň. Odtud plyne, že $q(x) = 0$, a tudíž $c(x) = m(x) \cdot g(x)$. □

Příklad 3.5.10 Mějme cyklický kód C tvořený slovy 0000,1010,0101 a 1111, který kóduje zprávy 00,10,01,11. Polynom $g(x) = 1 + x^2$. Kódová slova zapíšeme s využitím polynomu $g(x)$ následovně:

$$\begin{aligned} c_1 &= 0 \\ c_2 &= 1 + x^2 = g(x) \\ c_3 &= x + x^3 = xg(x) \\ c_4 &= 1 + x + x^2 + x^3 = (1 + x)g(x) \end{aligned}$$

Pomocí polynomu $g(x)$ odvodíme také dimenzi a velikost kódu.

Pokud je polynomu $g(x)$ stupně r, stupeň polynomu $m(x)$ je $\deg(m(x)) \leq n - r$. Proto existuje q^{n-r} možných polynomů, kterými můžeme zprávu zakódovat, tudíž existuje q^{n-r} kódových slov. Dimenze cyklického kódu je proto rovna $k = n - r$. Stupeň $g(x)$ můžeme zapsat jako $\deg(g(x)) = n - k$.

Z předchozích definic a vět vyplývá, že pro plné specifikování cyklického $[n, k]$-kódu stačí uvést pouze polynom $g(x)$, ze kterého jsme schopni odvodit všechna kódová slova. Polynom $g(x)$ se nazývá **generující polynom**.

Věta 3.5.11 [2] *Mějme generující polynom* $g(x) = g_0 + g_1x + ... + g_{r-1}x^{r-1} + x^r$ *, pak* g_0 *je nenulové.*

Důkaz Předpokládejme, že $g_0 = 0$, pak $g(x) = x(g_1 + g_2x + ... + g_{r-1}x^{r-2} + x^{r-1})$. Cyklickým posunem o $n - 1$ pozic dostaneme další kódové slovo:

$$g'(x) = x^{n-1}g(x) \mod (x^n - 1),$$

$$g'(x) = x^n(g_1 + g_2x + ... + g_{r-1}x^{r-2} + x^{r-1}) \mod (x^n - 1),$$

$$g'(x) = g_1 + g_2x + ... + g_{r-1}x^{r-2} + x^{r-1}.$$

Tím pádem je však stupeň $\deg(g'(x)) < r$, což je v rozporu s tím, že $g(x)$ je jediný monický polynom s nejmenším stupněm, tudíž $g_0 \neq 0$ □

Věta 3.5.12 [2] *Generující polynom* $g(x)$*, cyklického [n,k]-kódu C, je dělitelem polynomu* $(x^n - 1)$.

Důkaz Předpokládejme, že $g(x)$ nedělí $(x^n - 1)$. Pak polynom $(x^n - 1)$ zapíšeme jako $q(x)g(x) + r(x)$, kde stupeň zbytkového polynomu $r(x)$ je menší než stupeň $g(x)$. Neboť jak $q(x)g(x)$, tak $r(x)$ musí být kódová slova a současně $g(x)$ musí být nejmenšího stupně, musí být $r(x)$ rovno 0. Tudíž $(x^n - 1) = q(x)g(x)$, a proto $g(x)$ je dělitelem $(x^n - 1)$. □

Jelikož teď víme, že polynom $g(x)$ dělí $(x^n - 1)$, dostáváme, že $(x^n - 1) = h(x)g(x)$, kde $h(x) = h_0 + h_1x + ... + h_kx^k$ je polynom stupně k. Polynom $h(x)$ se nazývá **kontrolní polynom**. Jeho důležitou vlastností, která má význam při dekódování, je, že $h(x)g(x) = 0 \mod (x^n - 1)$. Zároveň odtud plyne, že každý dělitel polynomu $(x^n - 1)$ může generovat nějaký cyklický kód.

V tuto chvíli již máme všechny náležitosti a dostáváme se k samotné konstrukci kódu.

Polynom $m(x) = m_0 + m_1x + ... + m_{k-1}x^{k-1}$, reprezentující zprávu před zakódováním, můžeme jednoduše zakódovat vynásobením generujícím polynomem, jelikož výsledkem $m(x)g(x)$ je určitě kódové slovo stupně nejvýše $n - 1$.

Příklad 3.5.13 Mějme cyklický $[7, 4]$-kód C definovaný nad $GF(2)$, jehož generující polynom je stupně 3 a je dělitelem polynomu $(x^7 - 1)$. Jelikož polynom $(x^7 - 1)$ můžeme rozepsat jako $(x^7 - 1) = (1 + x)(1 + x + x^3)(1 + x^2 + x^3)$, máme dvě možnosti, jak kód zkonstruovat. Podle generujícího polynomu $g_1(x) = (1 + x + x^3)$ nebo $g_2(x) = (1 + x^2 + x^3)$. Následující tabulka uvádí konstrukci kódu pomocí g_1:

Zpráva $\vec{m}$	$m(x)$	$c(x) = m(x)g(x)$	Kódové slovo $\vec{c}$
0000	0	0	0000000
0001	x^3	$x^3 + x^4 + x^6$	0001101
0010	x^2	$x^2 + x^3 + x^5$	0011010
0011	$x^2 + x^3$	$x^2 + x^4 + x^5 + x^6$	0010111
0100	x	$x + x^2 + x^4$	0110100
0101	$x + x^3$	$x + x^2 + x^3 + x^6$	0111001
0110	$x + x^2$	$x + x^3 + x^4 + x^5$	0101110
0111	$x + x^2 + x^3$	$x + x^5 + x^6$	0100011
1000	1	$1 + x + x^3$	1101000
1001	$1 + x^3$	$1 + x + x^4 + x^6$	1100101
1010	$1 + x^2$	$1 + x + x^2 + x^5$	1110010
1011	$1 + x^2 + x^3$	$1 + x + x^2 + x^3 + x^4 + x^5 + x^6$	1111111
1100	$1 + x$	$1 + x^2 + x^3 + x^5$	1011010
1101	$1 + x + x^3$	$1 + x^2 + x^6$	1010001
1110	$1 + x + x^2$	$x^2 + x^3 + x^5$	0011010
1111	$1 + x + x^2 + x^3$	$1 + x^3 + x^5 + x^6$	1001011

Tabulka 12: Konstrukce cyklického $[7, 4]$-kódu s generujícím polynomem $g(x) = (1 + x + x^3)$

Tento zkonstruovaný kód má minimální vzdálenost $d = 3$ a je schopný opravit 1 chybu ($t = 1$).

Jelikož je cyklický kód zároveň i lineární, musí mít kromě generujícího polynomu také generující matici. Tu získáme poměrně jednoduše právě z generujícího polynomu.

Předpokládejme množinu k polynomů $g(x), xg(x), ..., x^{n-k}g(x)$. Tyto polynomy jsou lineárně nezávislé a současně představují kódová slova cyklického kódu o dimenzi $n-k$. Proto je můžeme použít ke konstrukci generující matice G o velikosti $k \times n$ následovně:

$$G(x) = \begin{bmatrix} g_0(x) = g(x) \\ g_1(x) = xg(x) \\ \vdots \\ g_{k-1}(x) = x^{k-1}g(x) \end{bmatrix} \iff G = \begin{bmatrix} g_0 & g_1 & \cdots & g_{n-k} & 0 & \cdots & 0 & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k} & 0 & \cdots & 0 \\ \vdots & & & & & & & \\ 0 & 0 & \cdots & 0 & g_0 & g_1 & \cdots & g_{n-k} \end{bmatrix},$$

kde $g_0 = g_{n-k} = 1$.

Příklad 3.5.14 Generující matice G našeho cyklického $[7, 4]$-kódu s generujícím polynomem $g(x) = (1 + x + x^3)$ je:

$$G(x) = \begin{bmatrix} g(x) \\ xg(x) \\ x^2g(x) \\ x^3g(x) \end{bmatrix} \iff G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Z generující matice jsme pak schopni získat i kontrolní matici H (viz. kapitola **3.4**).

Příklad 3.5.15 Kontrolní matice H našeho cyklického $[7, 4]$-kódu s generujícím polynomem $g(x) = (1 + x + x^3)$ je:

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

3.6 Kódování a dekódování cyklických kódů

Kódování cyklických kódů bylo již zmíněno v předchozí podkapitole při konstrukci (příklad **3.5.13**). Jednoduše vynásobíme polynom $m(x)$, reprezentující zprávu, s generujícím polynomem $g(x)$.

Dekódování cyklických kódů může probíhat stejně jako u lineárních kódů, protože cyklické kódy jsou lineární. Připomeňme, že při dekódování lineárních kódů jsme využívali řádkový vektor $S(\vec{r})$, představující tzv. *syndrom*, který splňoval rovnici $S(\vec{r}) = \vec{r}H^T$. Nic nám tedy nebrání na cyklické kódy pohlížet jen jako na lineární a sestavit syndromovou tabulku.

Příklad 3.6.1 Syndromová tabulka pro cyklický [7,4]-kód s generujícím polynomem $g(x) = (1 + x + x^3)$ vypadá takto:

Coset leadeři	Syndromy
0000000	000
0000001	001
0000010	010
0100000	011
0000100	100
0001000	101
1000000	110
0010000	111

Tabulka 13: Syndromová tabulka cyklického [7,4]-kódu

Analogicky ke dvojicím generující matice/generující polynom a kontrolní matice/kontrolní polynom, mají cyklické kódy také obdobu syndromu nazvanou **syndromový polynom**, který má následující podobu:

$$S(x) = s_0 + s_1x + , ..., + s_{n-k-1}x^{n-k-1}.$$

Definice 3.6.2 *Mějme cyklický kód C s generujícím polynomem g(x) a přijaté slovo r(x). Syndromovým polynomem nazveme zbytek po dělení polynomu r(x) generujícím polynomem g(x).*

Syndromový polynom tudíž splňuje následující rovnici:

$$S(x) = r(x) \mod (g(x)).$$

Polynom $r(x)$ reprezentuje přijaté slovo, a proto:

$$r(x) = m(x)g(x) + e(x),$$

kde $e(x)$ představuje chybu. Pokud je přijaté slovo kódovým slovem, chybový polynom $e(x) = 0$. Zároveň je jasné, že

$$m(x)g(x) = 0 \mod (g(x)).$$

Tím pádem ale můžeme psát:

$$S(x) = r(x) \mod (g(x)) = m(x)g(x) + e(x) \mod (g(x)) = e(x) \mod (g(x)).$$

Syndromový polynom $S(x)$ je roven 0 pouze tehdy, je-li chyba $e(x)$ rovna 0.

Poznámka Syndromový polynom **není** pouze polynomiálně reprezentovaný syndrom. Když si v cyklickém [7,4]-kódu s generujícím polynomem $g(x) = (1 + x + x^3)$ vybereme například vektor $\vec{v} = [0100000]$, jeho syndrom bude roven:

$$S(\vec{v}) = \vec{v}H^T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}.$$

Pokud ale budeme chtít spočítat syndromový polynom vektoru $\vec{v}$, jež reprezentujeme polynomem $v(x) = x$, dostaneme:

$$S(x) = v(x) \mod (g(x)) = x \mod (1 + x + x^3) = x,$$

což odpovídá vektoru [010] a nikoliv vektoru [011].

Velice důležitou vlastností syndromového polynomu je, že pokud polynomu $r(x)$ odpovídá syndromový polynom $S(x)$, pak posunutému polynomu $r^{(i)}(x)$ odpovídá posunutý syndromový polynom $S^{(i)}(x)$, který cyklicky posuneme takto:

$$S^{(i)}(x) = x^i \cdot S(x) \mod (g(x)).$$

Tato vlastnost nám umožňuje zredukovat syndromovou tabulku, protože pro jednoho coset leadera a všechny jeho cyklické posuny nám v paměti stačí uchovávat jediný syndromový polynom.

Příklad 3.6.3 Syndromovou tabulku z příkladu **3.6.1** zredukujeme na:

Coset leadeři	Syndromové polynomy
0000000	000
0000001	101

Tabulka 14: Redukovaná syndromová tabulka cyklického [7,4]-kódu

Všechny ostatní coset leadery a jejich syndromové polynomy jsme schopni získat cyklickými posuny. Zároveň bychom mohli z tabulky vypustit i nulový vektor.

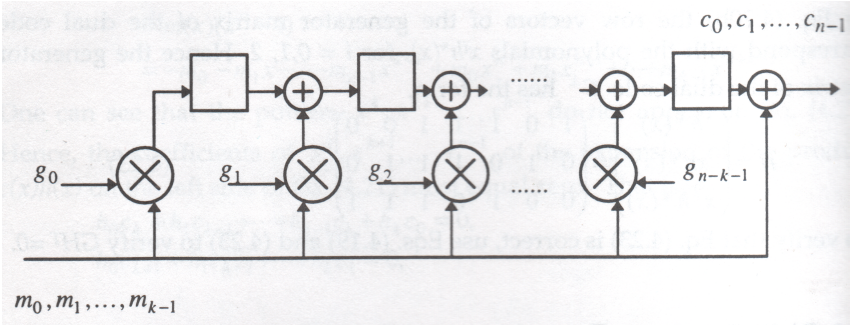
Nyní uveďme postup dekódování s redukovanou syndromovou tabulkou:

1. Nejprve spočteme syndromový polynom přijatého slova $\vec{r}$.
2. Vypočtený syndromový polynom se pokusíme najít v syndromové tabulce. Pokud se v tabulce nachází, nalezneme příšlušnou chybu a dekódujeme na $\vec{c} = \vec{r} - \vec{e}$.
3. Pokud se v tabulce nenachází, cyklicky syndromový polynom posuneme. Posouváme tak dlouho, dokud na příslušný syndromový polynom v tabulce nenarazíme. Přitom počítáme, kolikrát jsme již posunutí provedli.
4. Jakmile dostaneme syndromový polynom, který se v tabulce nachází, nalezneme příšlušnou chybu a tu cyklicky posuneme **opačným** směrem (cyklický posun opačným směrem vektoru o délce n můžeme chápat jako $n-1$ posunů standardním směrem) tolikrát, kolikrát jsme posunuli syndromový polynom.
5. Posunutou chybu použijeme k dekódování na $\vec{c} = \vec{r} - \vec{e}$.

Příklad 3.6.4 Mějme opět cyklický [7,4]-kód s generujícím polynomem $g(x) = (1 + x + x^3)$ a přijaté slovo $\vec{r} = [0110000]$, které odpovídá polynomu $r(x) = x + x^2$. Dekódujme nyní pomocí redukované syndromové tabulky z příkladu **3.6.3**:

1. Syndromový polynom přijatého slova $\vec{r}$ je $S(x) = x^2 + x$, což odpovídá vektoru [011].
2. Vektor [011] se v tabulce nenachází.
3. Syndrom $S(x) = x^2 + x$ jednou cyklicky posuneme na $S^{(1)}(x) = x(x^2 + x) \mod (1 + x + x^3) = x^2 + x + 1$, což odpovídá vektoru [111]. Ani tento vektor se v tabulce nenachází, a tak posouváme podruhé na $S^{(2)}(x) = x(x^2 + x + 1) \mod (1 + x + x^3) = x^2 + 1$, což odpovídá vektoru [101]. Tento vektor se již v tabulce nachází.
4. Vektor [101] koresponduje s coset leaderem [0000001]. Protože jsme ale museli syndromový polynom dvakrát posunout, musíme coset leadera dvakrát posunout v opačném směru, čímž získáme chybu $\vec{e} = [0000100]$.
5. Dekódujeme na $\vec{c} = \vec{r} - \vec{e} = [0110000] - [0000100] = [0110100]$.

Zmenšení syndromové tabulky není jediná výhoda cyklických kódů. Mnohem snazší je také konstrukce samotného kodéru a dekodéru, jelikož lze využít tzv. **posuvné registry** (*z angl. shift register*), které umí efektivně realizovat cyklické posuny. K realizaci posunů využívají posuvné registry klopných obvodů. Na následujícím obrázku (zdrojem obrázku je [2]) můžeme vidět kodér cyklického kódu:



Obrázek 1: Kodér cyklického kódu

Posuvný registr na obrázku kóduje součinem $c(x) = m(x)g(x)$. Kromě klopných obvodů, které jsou na uvedeném obrázku reprezentovány prázdnými čtverci, využívá také sčítaček a násobiček, což jsou obvody schopné provádět operace sčítání a násobení i v modulární aritmetice.

4 Reed-Solomonovy kódy

V této kapitole budeme zkoumat Reed-Solomonovy kódy. Rozebereme konstrukci, kódování, dekódování a na závěr zmíníme i konkrétní využití RS-kódů. Hlavním zdrojem informací pro tuto kapitolu je [2].

4.1 Konstrukce RS-kódu

Ke konstrukci, kódování a dekódování RS-kódů se využívá aritmetika konečných těles (viz kapitola **2**). Připomeňme, že konečná tělesa značíme GF(q) kde q je prvočíslo či mocnina prvočísla, představující počet prvků.

V konečném tělese se nachází primitivní prvek, který označme α. Mocniny primitivního prvku α (tzn. $1, \alpha, \alpha^2, ..., \alpha^{q-2}$) jsou různé a tvoří všech $(q-1)$ nenulových prvků GF(q). Primitivní prvek α je zároveň kořenem nějakého primitivního ireducibilního (dále nedělitelného) polynomu $p(x)$.

Nyní si ukážeme původní přístup ke konstrukci RS-kódů, se kterým přišli v roce 1958 *I. S. Reed* a *G. Solomon* [10], po nichž jsou Reed-Solomonovy kódy pojmenovány.

Zpráva $\vec{m} = (m_0, m_1, ..., m_{k-1}, m_k)$ je tvořena k symboly, jejichž abeceda představuje prvky GF(q). Zprávu můžeme také zapsat pomocí polynomu $m(x) = m_0 + m_1 x + ... + m_{k-2}x^{k-2} + m_{k-1}x^{k-1}$. Kódové slovo $\vec{c}$ pro zprávu $\vec{m}$ vytvoříme postupným dosazením všech q prvků konečného tělesa GF(q) do polynomu $m(x)$. Kódové slovo $\vec{c}$ vypadá následovně:

$$\vec{c} = (c_0, c_1, ..., c_{q-1}) = [m(0), m(\alpha), m(\alpha^2), ..., m(\alpha^{q-1})].$$

Když bude k symbolů zprávy postupně nabývat veškerých možných hodnot, dostaneme kompletní kód C:

$$C = \left\{ \vec{c} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{(q-1)} & \alpha^{2(q-1)} & \cdots & \alpha^{(k-1)(q-1)} \end{bmatrix} \cdot \vec{m}, \vec{m} \in [GF(q)]^k \right\}.$$

Takto vytvořený kód je lineární, protože lineární kombinace dvou libovolných zpráv délky k je jiná zpráva délky k. Počet symbolů k zprávy nazveme **dimenzí RS-kódu**, jelikož kódová slova tvoří vektorový prostor o dimenzi k ($V(k, n)$). Zároveň celková délka kódového slova n je rovna q, protože každé kódové slovo má q souřadnic. RS kódy se běžně specifikují právě dimenzí k a délkou n.

Jedná se o stejné parametry n a k, jako byly u obecného lineárního kódu. RS-kódy proto můžeme značit úplně stejně jako kódy lineární, tudíž jako $[n, k]$-kódy.

Původní přístup ke konstrukci nepočítal s faktem, že RS-kódy jsou i cyklické kódy a dají se tedy konstruovat pomocí generujícího polynomu. Objev nové metody vytváření RS-kódů pochází z roku 1961 a je připsán dvojici *Gorenstein* a *Zierler* [11]. V současnosti je tato metoda preferovanější.

RS-kódy vytvořené generujícím polynomem, které jsou definovány nad GF(q) mají délku $(q-1)$ (tzn. o jedna kratší než kód vytvořený původní metodou). Můžeme jej prodloužit na q či dokonce $(q+1)$, v takovém případě však ztrácíme cykličnost. Oblíbenými RS-kódy jsou kódy o délce 255, jelikož jsou definovány nad GF(256). Číslo 256 je možno zapsat jako 2^8, a tudíž každý prvek tělesa lze reprezentovat 8-bitovou sekvencí (jedním bytem).

Nyní popíšeme samotný postup konstrukce. Předpokládejme, že chceme sestrojit RS kód, definovaný nad GF(q), délky $(q-1)$, který je schopen opravit t chyb. Nenulové prvky GF(q) můžeme reprezentovat jako mocniny primitivního prvku α. Generující polynom má jako své kořeny $2t$ po sobě jdoucích mocnin prvku α. Generující polynom má tvar:

$$g(x) = \prod_{j=0}^{2t-1} (x - \alpha^{h+j}) = \sum_{j=0}^{2t-1} g_j x^j.$$

Konstanta h představuje tzv. *mocninu prvního kořene g(x)*. Pro různé hodnoty h pochopitelně dostáváme různé generující polynomy. Ve většině případů se volí $h = 1$. Volba konstanty ovlivňuje konstrukci kodéru a dekodéru.

Příklad 4.1.1 Sestrojme generující polynom pro RS [7,5]-kód, definovaný nad GF(2^3), kde primitivní polynom $p(x) = x^3 + x + 1$, $t = 1$ a konstanta $h = 1$. Využijme vzorec uvedený výše a dosaďme:

$$g(x) = \prod_{j=0}^{1} (x - \alpha^{h+j}) = (x - \alpha^1)(x - \alpha^2).$$

Roznásobíme závorky a dostaneme:

$$g(x) = x^2 - x(\alpha^2 + \alpha) + \alpha^3.$$

Ještě nejsme u konce, je třeba provést součet $(\alpha^2 + \alpha)$. Ke sčítání je potřebné prvek α vyjádřit nikoliv označením α, nýbrž jeho polynomiální reprezentací v GF(2^3). My však toto těleso máme kompletně sestrojeno v příkladě **2.2.10**, včetně tabulek sčítání a násobení pro prvky v polynomiální reprezentaci i reprezentaci pomocí prvku α.

S využitím tabulek můžeme vypočíst:

$$(\alpha^2 + \alpha) = x^2 + x = \alpha^4.$$

Výsledek dosadíme do $g(x)$. Zároveň se můžeme zbavit znamének mínus u koeficientů, jelikož v našem příkladě je (-1) kongruentní s 1. Finální podoba generujícího polynomu je:

$$g(x) = x^2 + \alpha^4 x + \alpha^3.$$

Pro demonstraci uveďme ještě jeden příklad na konstrukci generujícího polynomu pro větší kód.

Příklad 4.1.2 Vypočtěme generující polynom pro RS kód délky 31, schopný opravovat 2 chyby.

Parametry pro sestrojení kódu:

- Délka bloku n = 31
- Počet opravitelných chyb t = 2
- Konečné těleso GF(q)=GF($n + 1$)=GF(32)=GF(2^5);
- Délka zprávy k = n - 2t = $31 - 2 * 2 = 27$
- Ireducibilní polynom pro sestrojení tělesa $p(x) = x^5 + x^2 + 1$
- Konstanta h = 1

Přirazení koeficientu α:

α^1	x	α^{11}	$x^2 + x + 1$	α^{21}	$x^4 + x^3$
α^2	x^2	α^{12}	$x^3 + x^2 + x$	α^{22}	$x^4 + x^2 + 1$
α^3	x^3	α^{13}	$x^4 + x^3 + x^2$	α^{23}	$x^3 + x^2 + x + 1$
α^4	x^4	α^{14}	$x^4 + x^3 + x^2 + 1$	α^{24}	$x^4 + x^3 + x^2 + x$
α^5	$x^2 + 1$	α^{15}	$x^4 + x^3 + x^2 + x + 1$	α^{25}	$x^4 + x^3 + 1$
α^6	$x^3 + x$	α^{16}	$x^4 + x^3 + x + 1$	α^{26}	$x^4 + x^2 + x + 1$
α^7	$x^4 + x^2$	α^{17}	$x^4 + x + 1$	α^{27}	$x^3 + x + 1$
α^8	$x^3 + x^2 + 1$	α^{18}	$x + 1$	α^{28}	$x^4 + x^2 + x$
α^9	$x^4 + x^3 + x$	α^{19}	$x^2 + x$	α^{29}	$x^3 + 1$
α^{10}	$x^4 + 1$	α^{20}	$x^3 + x^2$	α^{30}	$x^4 + x$

Výpočet generujícího polynomu:

$$g(x) = \prod_{j=0}^{2t-1} (x - \alpha^{h+j}) = (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4) =$$

$$x^4 + x^3\alpha^4 + x^3\alpha^3 + x^2\alpha^7 + x^3\alpha^2 + x^2\alpha^6 + x\alpha^9 + x^3\alpha + x^2\alpha^4 + x\alpha^8 + x^2\alpha^3 + x\alpha^7 + x\alpha^6 + \alpha^{10} =$$

$$x^4 + x^3(\alpha^4 + \alpha^3 + \alpha^2 + \alpha) + x^2(\alpha^7 + \alpha^6 + \alpha^4 + \alpha^3) + x(\alpha^9 + \alpha^8 + \alpha^7 + \alpha^6) + \alpha^{10} =$$

$$x^4 + \alpha^{24}x^3 + \alpha^{19}x^2 + \alpha^{29}x + \alpha^{10}.$$

Nyní budeme zkoumat další vlastnosti RS-kódů.

Pokud by nás zajímalo, kolik kódových slov má například RS [7,5]-kód z příkladu **4.1.1**, stačí si uvědomit, že počet kódových slov je stejný jako počet zpráv, které lze zakódovat. Parametr $k = 5$ představuje délku zprávy. Máme tudíž pět pozic, které nabývají jakékoliv hodnoty z tělesa $\mathrm{GF}(2^3)$, nad kterým je kód definován. Počet možných zpráv je $2^{3^5} = 2^{15} = 32768$. Přesně tolik kódových slov obsahuje náš RS [7,5]-kód.

Stupeň kódového polynomu $c(x) = m(x)g(x)$ může být jakýkoliv v rozpětí od $2t$ po $(q-2)$. Kódový polynom stupně $(q-2)$ odpovídá kódovému vektoru o $(q-1)$ souřadnicích. Dimenze kódu s generujícím polynomem stupně $2t$ je $k = q - 2t - 1$. Odtud plyne, že RS kód délky $(q-1)$ a dimenze k, je schopen opravit t chyb, kde $t = \lfloor (q-k-1)/2 \rfloor$ ($\lfloor x \rfloor$ představuje tzv. dolní celou část).

Ze Singletonova odhadu (viz kapitola **3.2**) vyplývá, že schopnost opravit $\lfloor (q-k-1)/2 \rfloor$ chyb je nejlepší možná, jaké jsme schopni pro blokový kód našich parametrů dosáhnout. Tedy RS-kódy dosahují Singletonova odhadu a, jak už jsme dříve zmínili, spadají do třídy MDS kódů. Pomocí Singletonova odhadu si rovněž můžeme ověřit předchozí výpočet, který měl za cíl zjistit počet slov RS [7,5]-kódu definovaného nad $\mathrm{GF}(2^3)$. Kódy splňující Singletonův odhad musejí mít q^{n-d+1} kódových slov. Parametry $q = 2^3$ a $n = 7$ již známe. Potřebujeme určit minimální vzdálenost d. Protože se koeficienty dvou různých polynomů stupně nejvýše $k-1$ shodují v nejvýše $k-1$ složkách, liší se jakákoliv dvě kódová slova RS-kódu o alespoň $n-(k-1) = n-k+1$ souřadnic. Navíc však existují dva různé polynomy, jejichž koeficienty se shodují v $k-2$ složkách. Minimální vzdálenost RS-kódů je proto rovna $d = n-k+1$. Parametr $k = 5$ již ale také známe, můžeme tedy dosadit. Počet slov RS [7,5]-kódu podle Singletonova odhadu vychází $q^{n-(n-k+1)+1} = q^k = 2^{3^5} = 2^{15} = 32768$ čili stejně, jako při předchozím výpočtu.

Všechny kódové polynomy RS-kódu jsou násobky generujícího polynomu. To znamená, že jako kořeny musejí mít stejných $2t$ po sobě jdoucích mocnin primitivního prvku α, jako je tomu u generujícího polynomu $g(x)$. Tato vlastnost se využívá při dekódování. Pokud jsme přijali kódové slovo, musí být kořeny generujícího polynomu také kořeny přijatého slova.

Dalším z parametrů, kterým se obecné, a tudíž i Reed-Solomonovy, kódy popisují, je **informační poměr kódu**[2](*z angl. rate*). Označíme jej R a vypočteme jako $R = k/n$ (délka zprávy

děleno celková délka kódového slova). Informační poměr se nejčastěji udává v procentech a čím větší hodnoty dosahuje, tím menší má kód redundanci. Kódy s velkým informačním poměrem však často nemají příliš dobrou schopnost opravování chyb (viz kapitola **3.2**).

Příklad 4.1.3 Informační poměr RS [7,5]-kódu z příkladu **4.1.1** je $R = k/n = 5/7 \doteq 71,429\ \%$ $(t = 1)$. Pro RS [31,27]-kód z příkladu **4.1.2** je $R = k/n = 27/31 \doteq 87,097\ \%$ $(t = 2)$.

4.2 Kódování a dekódování RS-kódů

Kódování RS-kódů, které zde nyní uvedeme, vychází z konstrukce RS-kódů jakožto cyklických kódů. Požadovaný kódový polynom $c(x)$ máme opět možnost získat prostým vynásobením polynomu $m(x)$, představujícího zprávu, s generujícím polynomem $g(x)$.

Příklad 4.2.1 Mějme RS [7,5]-kód nad GF(2^3) z příkladu **4.1.1**. Vypočetli jsme, že tento kód má pro konstantu $h = 1$ generující polynom $g(x) = x^2 + \alpha^4 x + \alpha^3$. Nyní zakódujme zprávu $m(x) = \alpha^4 x^2 + \alpha^3 x$:

$$c(x) = m(x)g(x) = (\alpha^4 x^2 + \alpha^3 x) \cdot (x^2 + \alpha^4 x + \alpha^3),$$

$$c(x) = \alpha^4 x^4 + (\alpha^4\alpha^4)x^3 + (\alpha^4\alpha^3)x^2 + \alpha^3 x^3 + (\alpha^3\alpha^4)x^2 + (\alpha^3\alpha^3)x,$$

$$c(x) = \alpha^4 x^4 + \alpha x^3 + x^2 + \alpha^3 x^3 + x^2 + \alpha^6 x,$$

$$c(x) = \alpha^4 x^4 + x^3 + \alpha^6 x.$$

V praxi se ovšem používá i mírně upravená varianta kódování, kdy zprávu nejprve cyklicky posuneme a následně vydělíme modulo generující polynom. Přesný postup tohoto kódování je následující:

1. Cyklicky posuňme polynom $m(x)$ tím, že jej vynásobíme s polynomem x^{d-1}.
2. Získejme polynom $p(x)$ tak, že $p(x) = x^{d-1}m(x) \mod g(x)$.
3. Kódové slovo $c(x)$ dostaneme jako součet $c(x) = x^{d-1}m(x) + p(x)$.

Příklad 4.2.2 Vezměme si opět RS [7,5]-kód nad GF(2^3), kde $g(x) = x^2+\alpha^4 x+\alpha^3$. Připomeňme ještě, že minimální vzdálenost d tohoto kódu je $d = n - k + 1 = 7 - 5 + 1 = 3$. Zakódujme nyní zprávu $m(x) = \alpha^4 x^2 + \alpha^3 x$:

1. Součin $x^{d-1}m(x) = x^2 \cdot (\alpha^4 x^2 + \alpha^3 x) = \alpha^4 x^4 + \alpha^3 x^3$.
2. Polynom $p(x) = x^{d-1}m(x) \mod g(x) = \alpha^4 x^4 + \alpha^3 x^3 \mod (x^2 + \alpha^4 x + \alpha^3) = \alpha^5 x + \alpha$.
3. Kódové slovo $c(x) = x^{d-1}m(x) + p(x) = \alpha^4 x^4 + \alpha^3 x^3 + \alpha^5 x + \alpha$.

Tímto kódováním zařídíme, že symboly, které se vyskytovaly ve zprávě, budou v nezměněné podobě i v kódovém slově (takovémuto kódování se říká **systematické**). V našem příkladě se jednalo o symboly α^4 a α^3.

Dva uvedené způsoby kódování nejsou jediné. Pro další kódovací algoritmy můžeme čtenáře odkázat na zdroj [2].

Dekódování RS-kódů můžeme rozčlenit do dvou kategorií. Klasické kódování, kdy došlo k chybě a je třeba ji opravit, a kódování, kdy některé ze symbolů vypadly úplně a nelze je vůbec přečíst. My se zaměříme hlavně na opravu chyb. Opravu nečitelných symbolů pouze zmíníme.

Stejně jako u obecných cyklických kódů můžeme polynom $r(x)$, reprezentující přijaté slovo, definovat jako součet kódového polynomu $c(x)$ a chybového polynomu $e(x)$. Při dekódování využíváme faktu, že polynom $c(x)$ je dělitelný generujícím polynomem $g(x)$ a také, že polynom $g(x)$ má jako své kořeny $2t$ ($2t = d-1$) po sobě jdoucích mocnin primitivního prvku α. Pokud do kódového polynomu $c(x)$ dosadíme kořeny polynomu $g(x)$, dostáváme ze zmíněných vlastností vztah:

$$c(\alpha^i) = m(\alpha^i)g(\alpha^i) = 0,\ i = 1, 2, ..., d-1.$$

Kořeny polynomu $g(x)$ nyní dosadíme také do přijatého slova $r(x)$:

$$r(\alpha^i) = c(\alpha^i) + e(\alpha^i),$$

$$r(\alpha^i) = e(\alpha^i),$$

$$r(\alpha^i) = \sum_{j=0}^{n-1} e_j \alpha^{ij},\ i = 1, 2, ..., d-1.$$

Vidíme, že po dosazení kořenů generujícího polynomu do přijatého slova zůstanou nenulové pouze složky chyby a nikoliv kódového slova. Tato vlastnost nápadně připomíná vlastnost syndromového polynomu, definovaného u cyklických kódů v kapitole **3.6**. U RS-kódů ovšem polynomy reprezentujeme mocninou primitivního prvku α, a proto používáme, stejně jako u lineárních kódů, označení **syndrom** a nikoliv syndromový polynom. Nadefinujme nyní syndrom RS-kódu.

Definice 4.2.3 *Mějme Reed-Solomonův $[n, k]$-kód definovaný nad GF(q) a přijaté slovo $r(x)$. Syndromem S_i RS-kódu se nazývá výsledek po dosazení i-té mocniny primitivního prvku α^i, kde $i \in \{1, ..., d-1\}$, do přijatého slova $r(x)$.*

Syndrom RS-kódu je dán vztahem:

$$S_i = r(\alpha^i) = \sum_{j=0}^{n-1} r_j \alpha^{ij},\ i = 1, 2, ..., d-1.$$

Poznámka Ve výše uvedeném vztahu si můžeme všimnout, že narozdíl od syndromu definovaného u obecných lineárních kódů, kde jednomu přijatému slovu odpovídal jeden syndrom, má jedno přijaté slovo RS-kódu více syndromů.

Příklad 4.2.4 Mějme RS [7,5]-kód z příkladu **4.1.1** a přijaté slovo $r(x) = \alpha^3 x^4 + \alpha^2 x^3 + \alpha^6 x + 1$. Tento kód má generující polynom se dvěma kořeny α a α^2, proto bude mít přijaté slovo 2 syndromy:

$$S_1 = r(\alpha^1) = \alpha^3(\alpha^1)^4 + \alpha^2(\alpha^1)^3 + \alpha^6(\alpha^1) + 1 = 1 + \alpha^5 + 1 + 1 = \alpha^4,$$

$$S_2 = r(\alpha^2) = \alpha^3(\alpha^2)^4 + \alpha^2(\alpha^2)^3 + \alpha^6(\alpha^2) + 1 = \alpha^4 + \alpha + \alpha + 1 = \alpha^5.$$

Nyní s pomocí syndromů budeme schopni získat chybu $e(x)$. Předpokládejme, že na neznámých pozicích $i_1, i_2, ..., i_v$ nastalo v chyb ($0 \leq v \leq t$). Chyba bude mít následující tvar:

$$e(x) = e_{i_1} x^{i_1} + e_{i_2} x^{i_2} + ... + e_{i_1} v^{i_v}$$

Neznáme pozice, hodnoty ani počet chyb. K odhalení chybového polynomu však všechny tyto parametry potřebujeme znát. Než se budeme zabývat jejich nalezením, zavedeme (podle [2]) nové označení. Hodnoty chyb označme jako

$$e_{i_l} = Y_l,\ l = 1, 2, ..., v$$

a pozice chyb po dosazení primitivního prvku jako

$$\alpha^{i_l} = X_l,\ l = 1, 2, ..., v.$$

Pokud nové značení dosadíme do rovnice $S_i = e(\alpha^i)$, dostáváme soustavu nelineárních rovnic:

$$S_i = Y_1 X_1^i + Y_2 X_2^i + ... + Y_v X_v^i,\ i = 1, 2, ..., d-1. \quad (1)$$

Dekódovací algoritmus by se dal shrnout do tří kroků:

1. Spočteme syndromy přijatého slova $r(x)$.
2. Vyřešíme soustavu $d-1$ nelineárních rovnic a tím určíme chybu $e(x)$.
3. Dekódujeme na kódové slovo $c(x) = r(x) - e(x)$.

Největším problémem je pochopitelně vyřešení soustavy nelineárních rovnic. Tím, jak řešení docílit, se budeme zabývat nyní.

Úlohu je výhodné rozdělit na další mezikroky. Nadefinujme proto tzv. **polynom pro lokalizaci chyb** (*z angl. error-locator polynomial*).

Definice 4.2.5 *Mějme soustavu nelineárních rovnic pro nalezení chybového polynomu* $S_i = Y_1X_1^i + Y_2X_2^i + ... + Y_vX_v^i$, *kde* $i = 1, 2, ..., d-1$ *a* v *představuje počet nastalých chyb. Pak polynomem pro lokalizaci chyb* $\Lambda(x)$ *nazveme polynom stupně* v, *jehož kořeny jsou prvky inverzní k prvkům* X_l, *reprezentujícím pozice chyb.*

Polynom pro lokalizaci chyb má tvar:

$$\Lambda(x) = \prod_{l=1}^{v}(1 - xX_l),\ X_l = \alpha^{i_l}.$$

K nalezení polynomu pro lokalizaci chyb se využívá například **Petersonův algoritmus** nebo **Berlekamp-Masseyho algoritmus**. Oba algoritmy jsou poměrně detailně popsány v [2] nebo v původních zdrojích [12] a [13].

Po nalezení polynomu pro lokalizaci chyb určíme jeho kořeny. Určení kořenů se provádí tzv. **Chienovým vyhledáváním** [14] (*z angl. Chien search*). Jedná se o algoritmus, kdy hrubou silou zkoušíme dosazovat všechny nenulové prvky konečného tělesa. Po nalezení kořenů určíme jejich inverze a tak získáme požadované pozice chyb.

Příklad 4.2.6 Znovu využijeme RS-[7,5] kód a polynom pro lokalizaci chyb $\Lambda(x) = \alpha x + 1$. Postupným dosazováním všech nenulových prvků GF(2^3) provedeme Chienovo vyhledávání:

$$\begin{gathered}\Lambda(1) = \alpha^3 \\ \Lambda(\alpha) = \alpha^6 \\ \Lambda(\alpha^2) = \alpha \\ \Lambda(\alpha^3) = \alpha^5 \\ \Lambda(\alpha^4) = \alpha^4 \\ \Lambda(\alpha^5) = \alpha^2 \\ \Lambda(\alpha^6) = 0\end{gathered}$$

Zjistili jsme, že kořenem polynomu pro lokalizaci chyb je prvek α^6. Prvek inverzní k prvku α^6 je prvek α, protože v GF(2^3) platí, že $\alpha^6 \cdot \alpha = 1$. Pozice chyby $X_1 = \alpha$.

Když máme nalezeny pozice chyb, můžeme začít hledat i hodnoty. Stejně jako u hledání polynomu pro lokalizaci chyb máme více možných způsobů realizace.

Nejjednodušším řešením z hlediska matematického aparátu je zpětné dosazení nalezených pozic do soustavy (1), čímž z nelineární soustavy dostáváme soustavu lineární.

Příklad 4.2.7 Pokračujme s RS [7,5]-kódem, kde jsme v příkladě **4.2.7** zjistili, že chyba nastala na pozici $X_1 = \alpha$. Dále z příkladu **4.2.4** známe syndromy přijatého slova $r(x) = \alpha^3x^4 + \alpha^2x^3 + \alpha^6x + 1$ ($S_1 = \alpha^4$, $S_2 = \alpha^5$). Dosadíme do soustavy (1):

$$S_1 = Y_1X_1 = Y_1\alpha = \alpha^4$$
$$S_2 = Y_1X_1 = Y_1\alpha^2 = \alpha^5$$

Snadno dopočteme, že koeficient $Y_1 = \alpha^3$. Nyní ovšem máme vše potřebné k sestavení chybového polynomu. Víme, že pozice chyby je $X_1 = \alpha^1$ (což znamená, že se nachází u x^1) a hodnota chyby je $Y_1 = \alpha^3$. Chybový polynom $e(x)$ je proto α^3x. Můžeme dekódovat na kódové slovo:

$$c(x) = r(x) - e(x) = (\alpha^3x^4 + \alpha^2x^3 + \alpha^6x + 1) - (\alpha^3x) = \alpha^3x^4 + \alpha^2x^3 + \alpha^4x + 1.$$

V tomto případě bylo vyřešení soustavy (v podstatě se ani nejednalo o soustavu, k výpočtu by nám postačila pouze jedna rovnice) velice snadné, jelikož došlo k jediné chybě. Pro větší počet nastalých chyb je vyřešení soustavy pochopitelně pracnější. Demonstrujme nyní případ, kdy nastaly tři chyby.

Příklad 4.2.8 Uvažujme RS [15,9]-kód, definovaný nad GF(2^4), kde operace jsou prováděny modulo ireducibilní polynom $p(x) = x^4 + x + 1$. Pro úplnost uveďme, že kód je schopen opravit tři chyby ($t = 3$), minimální vzdálenost $d = 7$ a generující polynom

$$g(x) = x^6 + \alpha^{10}x^5 + \alpha^{14}x^4 + \alpha^4x^3 + \alpha^6x^2 + \alpha^9x + \alpha^6.$$

Přijato bylo slovo

$$r(x) = x^9 + \alpha^{14}x^8 + \alpha^6x^7 + \alpha^2x^6 + \alpha^6x^5 + \alpha^{13}x^4 + \alpha^{12}x^3 + \alpha^7x^2 + \alpha^6x + \alpha^5.$$

Syndromy tohoto slova jsou

$$\begin{array}{l|l} S_1 = r(\alpha) = \alpha^3 & S_4 = r(\alpha^3) = \alpha^{14} \\ S_2 = r(\alpha^1) = \alpha^5 & S_5 = r(\alpha^4) = \alpha^7 \\ S_3 = r(\alpha^2) = \alpha^{10} & S_6 = r(\alpha^5) = \alpha \end{array}$$

Dále bylo zjištěno, že polynom pro lokalizaci chyb je $\Lambda(x) = \alpha^5x^3 + \alpha^9x^2 + \alpha^{13}x + 1$. Po provedení Chienova vyhledávání (po dosazení všech prvků GF(2^4)) dostáváme, že kořeny polynomu $\Lambda(x)$ jsou 1, α^{12} a α^{13}. Inverzní prvky k těmto kořenům a zároveň pozice chyb jsou pak $X_1 = 1 = \alpha^0$, $X_2 = \alpha^3$ a $X_3 = \alpha^2$. Sestavme nyní soustavu lineárních rovnic (1):

$$Y_1 + \alpha^3 Y_2 + \alpha^2 Y_3 = \alpha^3$$
$$Y_1 + \alpha^6 Y_2 + \alpha^4 Y_3 = \alpha^5$$
$$Y_1 + \alpha^9 Y_2 + \alpha^6 Y_3 = \alpha^{10}$$
$$Y_1 + \alpha^{12} Y_2 + \alpha^8 Y_3 = \alpha^{14}$$
$$Y_1 + Y_2 + \alpha^{10} Y_3 = \alpha^7$$
$$Y_1 + \alpha^3 Y_2 + \alpha^{12} Y_3 = \alpha$$

Jelikož máme tři neznámé, potřebujeme pouze tři libovolné rovnice z uvedených šesti. Vyberme například první tři. Soustavu vyřešíme pomocí Cramerova pravidla [8]:

$$Y_1 = \frac{\det\begin{vmatrix} \alpha^3 & \alpha^3 & \alpha^2 \\ \alpha^5 & \alpha^6 & \alpha^4 \\ \alpha^{10} & \alpha^9 & \alpha^6 \end{vmatrix}}{\det\begin{vmatrix} 1 & \alpha^3 & \alpha^2 \\ 1 & \alpha^6 & \alpha^4 \\ 1 & \alpha^9 & \alpha^6 \end{vmatrix}} = \frac{\alpha^2}{\alpha^3} = \alpha^{14},\ Y_2 = \frac{\det\begin{vmatrix} 1 & \alpha^3 & \alpha^2 \\ 1 & \alpha^5 & \alpha^4 \\ 1 & \alpha^{10} & \alpha^6 \end{vmatrix}}{\det\begin{vmatrix} 1 & \alpha^3 & \alpha^2 \\ 1 & \alpha^6 & \alpha^4 \\ 1 & \alpha^9 & \alpha^6 \end{vmatrix}} = \frac{\alpha}{\alpha^3} = \alpha^{13},$$

$$Y_3 = \frac{\det\begin{vmatrix} 1 & \alpha^3 & \alpha^3 \\ 1 & \alpha^6 & \alpha^5 \\ 1 & \alpha^9 & \alpha^{10} \end{vmatrix}}{\det\begin{vmatrix} 1 & \alpha^3 & \alpha^2 \\ 1 & \alpha^6 & \alpha^4 \\ 1 & \alpha^9 & \alpha^6 \end{vmatrix}} = \frac{\alpha^5}{\alpha^3} = \alpha^2.$$

Chybový polynom $e(x)$ má chyby o hodnotách α^{14}, α^{13} a α^2 na pozicích x^0, x^3 a x^2. Jeho tvar odpovídá $e(x) = \alpha^{13}x^3 + \alpha^2 x^2 + \alpha^{14}$. Dekódujeme na kódové slovo

$$c(x) = r(x) - e(x) = \alpha x^9 + \alpha^{14}x^8 + \alpha^6 x^7 + \alpha^2 x^6 + \alpha^6 x^5 + \alpha^{13}x^4 + \alpha x^3 + \alpha^{12}x^2 + \alpha^6 x + \alpha^{12}.$$

Ačkoliv je tento způsob hledání hodnot chyb jednoduchý z hlediska vyžadovaného matematického aparátu, je poměrně náročný pro dekodéry z hlediska požadovaného výpočetního výkonu. Existují další metody jako například **Forneyho algoritmus** a jeho upravená verze, které jsou mnohem jednodušší pro dekodér, stojí za nimi však výrazně složitější matematický aparát. V této práci je nebudeme rozebírat. Pokud by měl čtenář zájem, jsou rozebrány v [2] a [15]. Ve stejné publikaci je také rozebráno dekódování zpráv s nečitelnými symboly. K tomuto dekódování se využívá například **Euklidův algoritmus**.

Proveďme nyní celkové shrnutí a zhodnocení dekódovacích alogritmů RS-kódů. Narozdíl od dekódování obecných lineárních kódů umožnují RS-kódy mnohem efektivnější práci s většími kódy. Jak už bylo řečeno dříve, v praxi se často využívají kódy o délce 255. Dekódovat takto dlouhý kód jako obecný lineární kód by vyžadovalo enormní množství paměti a bylo mnohem pomalejší. Tím bychom podstatně omezili pole využitelnosti. Samotná konstrukce dekodérů je taktéž zjednodušena, jelikož se stejně jako u obecného cyklického kódu využívají posuvné registry.

Díky svým vlastnostem a efektivním kódovácím a dekódovacím algoritmům jsou RS-kódy v dnešní době hojně využívany. Konkrétní aplikace rozebereme v následující kapitole.

4.3 Aplikace RS-kódů

Samoopravné kódy jako takové dnes nacházejí své uplatnění prakticky všude, kde se nějakým způsobem přenášejí data. Každý kód se podle svých vlastností hodí do jiných situací. Jednou z nejpoužívanějších tříd kódů jsou právě Reed-Solomonovy kódy, jejichž konkrétními aplikacemi se budeme v této podkapitole zabývat.

RS-kódy pokrývají široké spektrum různých využití. Setkat se s nimi můžeme nejen u zařízení pro ukládání dat, jako jsou CD, DVD a Blu-Ray disky, ale také u čárových kódů, bezdrátových sítí, digitální televize a satelitní komunikace, včetně přenosu dat z vesmíru. Některé z těchto využití nyní rozebereme detailněji.

4.3.1 CCSDS

Consulative Committee for Space Data Systems (CCSDS), je celosvětové mezinárodní fórum, založené roku 1982, jež seskupuje vesmírné agentury, aby vytvářelo komunikační standardy pro vesmírné lety (oficiální stránky organizace: [16]).

V květnu 1984 tato organizace vydala oficiální standard [17], jež pro komunikační kanály vesmírných plavidel doporučuje využití RS [255,223]-kódu, definovaného nad GF(2^8). Operace jsou v tomto konečném tělese prováděny modulo ireducibilní polynom $p(x) = x^8 + x^7 + x^2 + x + 1$. Generující polynom tohoto kódu má tvar:

$$g(x) = \prod_{j=112}^{143} (x - \alpha^{11j}).$$

Konkrétně tento standard a s ním i tento kód využily například mise Ulysses (1990), Mars observer (1992), Pathfinder (1996) a Cassini (1997).

Je jasné, že standard vydaný v roce 1984 dnes již není zcela aktuální. Zajímavé však je, že skutečně aktuální komunikační standard ze září 2017 [18], se od toho původního příliš neliší. Stále se v něm objevuje RS-kód definovaný nad GF(2^8) s ireducibilním polynomem $p(x) = x^8 + x^7 + x^2 + x + 1$. Generující polynom má tvar:

$$g(x) = \prod_{j=128-t}^{127+t} (x - \alpha^{11j}),$$

kde t představuje počet opravitelných chyb. Standard doporučuje volbu parametru t jako 8 nebo 16. Můžeme si všimnout, že pokud zvolíme $t = 16$, dostaneme úplně stejný RS [255,223]-kód jako ve standardu z roku 1984. Pokud zvolíme $t = 8$, dostaneme RS [255,239]-kód.

4.3.2 CIRC

Dnes již nepříliš využívaná, nicméně velice důležitá aplikace RS-kódu je standard *Cross-interleaved Reed–Solomon coding* (CIRC) [2]. Jedná se o standard využívaný na kompaktních audio discích (Audio CD). CIRC popisuje samotný kód a nikoliv proces dekódování. Jak jsou informace uložené na disku dekódovány záleží na výrobci dané mechaniky, která provádí čtení.

Chyby jsou u CD způsobeny řadou faktorů. Jmenovitě se může jednat o bubliny vzduchu obsažené v plastu, nečistoty, otisky prstů a škrábance. Pochopitelně se dá předpokládat, že například škrábanec nebude tak malý, aby způsobil chybu pouze u jediného čteného bitu. Chyby u CD proto velice pravděpodobně budou nastávat v dávkách. Předpokládejme, že v přijatém slově nastalo 8 po sobě jdoucích chyb. Pokud budeme 8-bitové řetězce (1 bajt) reprezentovat jako jediný symbol, bude pro nás chyba v 8 bitech znamenat maximálně dva chybné symboly. A protože v konečných tělesech lze dlouhé polynomy reprezentovat jedinou mocninou primitivního prvku (jediným symbolem), jsou pro tento typ chyby ideální právě RS-kódy.

RS-kódy nejsou jediným prostředkem k ošetřování chyb, které se u CD využívají. Využívá se i tzv. **prokládání** (*z angl. interleaving*). Princip prokládání spočívá v ropztýlení chyb pro snazší čtení. Demonstrujme na příkladě.

Příklad 4.3.2.1 Pokud odesíláme například větu **TotoJePříkladProkládání** a nastanou čtyři po sobě jdoucí chyby, získáme větu **TotoJePříklad_____ládání**. Velice těžko se nám indentifikuje, jak zněla původní věta. Pokud však symboly v původní větě "popřeházíme", získáme například řetězec **TJoeřlPodlntríkPoaákáíd**. Nyní nastanou čtyři po sobě jdoucí chyby a řetězec se změní na **TJ_____lPodlntríkPoaákáíd**. Po přerovnání znaků nazpět dostaneme **Tot__J__P__íkladProk__ádání**, což je mnohem snazší přečíst.

Standard CIRC využívá pro své účely dvou RS-kódů C_1 a C_2, které jsou definovány nad GF(2^8), a tudíž používají již zmíněné 8-bitové řetězce. Za normálních okolností bychom očekávali, že kódy budou mít vzhledem k použitému konečnému tělesu délku $n = q - 1 = 255$. Takto dlouhý kód by však vyžadoval komplikovaný a hlavně drahý dekodér, o což výrobci CD přehrávačů pochopitelně nestojí. Kódy jsou proto poměrně razantně zkráceny a kód C_1 je [28,24]-kód a C_2 [32,28]-kód. Oba kódy mají minimální vzdálenost $d = 5$.

Jeden tzv. hudební **vzorek** (*z angl. sample*) má velikost 16 bitů. To koresponduje s 8-bitovými řetezci (k zakódování jednoho vzorku potřebujeme dva symboly). Vzorků se kóduje 12 najednou, čímž po přidání 4 symbolů redundance vytvoříme požadované kódové slovo kódu C_1 o délce 28 symbolů. Poté se provede prokládání a výsledný řetězec se zakóduje pomocí kódu C_2 na délku 32 symbolů.

V krátkosti také můžeme zmínit, že nástupce CD *Digital Versatile Disc* (DVD), využívá obdobně jako CIRC dvou RS-kódů C_1 ([182,172]-kód) a C_2 ([208,192]-kód), definovaných opět nad GF(2^8). Hlavním rozdílem ale je, že oba kódy nejsou využity za sebou, nýbrž zkombinovány do tzv. **Reed-Solomonova součinového kódu** (*z angl. Reed-Solomon Product Code*). Součinové kódy jsou nastíněny v [2]. Obdobně také Blu-ray disky (BD) využívají součinové RS-kódy.

4.3.3 QR-kódy

V současné době velice populárním příkladem čárového kódu je tzv. *Quick Response kód* (QR-kód) [19], který pochází z roku 1994. Původně byl vytvořen čistě pro automobilový průmysl v Japonsku, ale díky své rychlé čitelnosti a schopnosti uchovávat mnohem větší data než jeho předchůdci, si poměrně rychle získal přízeň i mimo něj. QR-kód většinou sestává z černých čtverců na bílém pozadí (barva čtverců ale může být i libovolná jiná), které jsou uspořádány do čtvercové mřížky. Výsledný obrazec je následně naskenován (například fotoaparátem) a poté je zpracována samotná informace, kterou QR-kód nese.

Při skenování kódu pochopitelně mohou nastávat chyby, ať už způsobené skenovacím zařízením, či nedostatky na samotném QR-kódu (kód na vizitce například může být promočený, špinavý, případně samotná vizitka může být natržená). Přesto lze data pořád korektně přečíst. K ošetření chyb QR-kódy využívají RS-kódy v kombinaci s prokládáním.

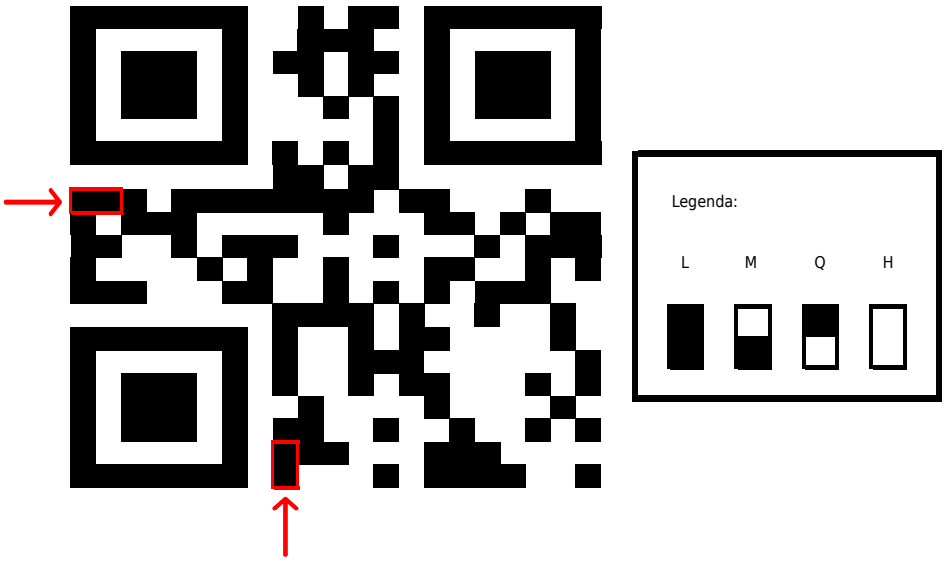
Schopnost opravovat chyby je u QR-kódů rozdělena na čtyři základní stupně podle toho, kolik procent poškozené informace je kód schopen opravit. Je jasné, že čím více procent opravíme, tím přidáme více redundantních symbolů a tím méně místa zbývá pro samotná data. Pokud chceme uchovávat více dat, potřebujeme větší (ve smyslu skutečné velikosti obrazce) QR-kódy. Stupně oprav uvádí následující tabulka:

Stupeň opravy	**Schopnost opravit chyby**
L (low)	až 7 %
M (medium)	až 15 %
Q (quality)	až 25 %
H (high)	až 30 %

Tabulka 15: Stupně oprav QR-kódu

Běžně používané jsou stupně L a M. Stupně Q a H jsou využívány v průmyslových zónách, kde je problém uchovat QR-kódy v čistotě a nepoškozené.

Informace o tom, jaký stupeň je u daného kódu použit, se nachází vlevo u dvou velkých čtverců, pomáhajících skenovacímu zařízení k lepší orientaci. U každého čtverce se nachází tatáž informace, jako pojistka, v případě nečitelnosti jedné z nich. Přesné pozice uvádí následující obrázek:



Obrázek 2: Pozice stupně opravy QR-kódu

Podle legendy vidíme, že QR-kód na obrázku používá ochranu proti chybám stupně L (pořadí čtení menších čtverců, reprezentujících stupeň, bereme ve směru červených šipek). Měl by být proto schopen opravit poškození až 7%.

Díky RS-kódům lze také provádět "umělecké zásahy", které mají za cíl přivést na sebe pozornost. Některé firmy například mohou do QR-kódu, jež nese odkaz na jejich oficiální stránky, zakomponovat své logo či jiný obrázek, čímž upoutají potenciálního zákazníka.

5 Závěr

Cílem této bakalářské práce bylo srozumitelné přiblížení a shrnutí principů fungování samoopravných kódů se zaměřením na vybranou skupinu kódů. Pro lepší porozumění problematiky jsme v práci rovněž uvedli demonstrace na konkrétních příkladech. Jako hlavní předmět práce byla zvolena podtřída blokových lineárních kódů, Reed-Solomonovy kódy. Zvolena byla hlavně pro široké spektrum využití prakticky ve všech oblastech, kde se nějakým způsobem přenášejí data.

Pro umožnění práce s Reed-Solomonovými kódy bylo nejprve nutné přiblížit jednu ze základních algebraických struktur, a to konečné těleso. Popsali jsme konstrukci této struktury, její vlastnosti a věnovali se modulární aritmetice, která se při počítání v konečných tělesech používá. Konkrétně jsou pro práci s Reed-Solomonovými kódy důležitá tělesa o velikosti mocniny prvočísla, kde jsou prvky reprezentovány jako polynomy. Poněkud problémové je hledání tzv. ireducibilního polynomu, který se při konstrukci takovýchto těles využívá. Různé metody ověřování, zda je daný polynom ireducibilní, jsou velmi pracné už pro poměrně malé stupně polynomů nebo využívají velice složitý matematický aparát.

Poté jsme přešli k samotným samoopravným kódům. Kromě vysvětlení pojmu jako takového, jsme uvedli vlastnosti, kterými se dají kódy odlišovat a popisovat. Zjistili jsme například co znamená, že je daný kód silnější než jiný. Dále jsme rozebrali tzv. hlavní problém teorie kódování. Tento problém spočívá v hledání co nejefektivnějšího kódu s co nejlepšími parametry v závislosti na jiných předem zvolených parametrech.

Jelikož jsou Reed-Solomonovy kódy podtřídou lineárních kódů, přiblížili jsme následně tuto skupinu kódů. Kódová slova lineárního kódu tvoří vektorový podprostor nějakého vektorového prostoru sestrojeného nad konečným tělesem. Pro práci s nimi se tedy využívá aparátu lineární algebry. Dále jsme se dozvěděli, že ačkoliv jsou konstrukce, kódování a dekódování této skupiny kódů velice jednoduché, prakticky mají smysl pouze pro malé kódy, jelikož dekódovací tabulka i redukovaná syndromová tabulka jsou příliš velké na to, aby byly uchovávány v paměti a efektivně využívány k rychlému dekódovacímu procesu. Pro obě tabulky jsme rovněž uvedli srovnání z hlediska velikosti vyžadované paměti.

Další podtřídou kódů, do které Reed-Solomonovy kódy spadají a jejíž vlastnosti můžeme pro práci s nimi využít, jsou cyklické kódy. Zjistili jsme, že namísto vektorové reprezentace zpráv a kódových slov je pro tuto třídu výhodnější reprezentace polynomiální. Polynomiální reprezentací získáme nové možnosti manipulace, konstrukce a hlavně kódování a dekódování. Při dekódování jsme se dozvěděli, že kromě redukování dekódovací tabulky cyklické kódy rovněž umožňují efektivnější konstrukci samotných kodérů a dekodérů s využitím tzv. posuvných registrů.

Všechny poznatky o lineárních a cyklických kódech jsme zúročili v kapitole o Reed-Solomonových kódech, jejichž srozumitelná prezentace byla hlavním cílem naší práce. Reed-Solomonovy kódy díky využití aritmetiky konečných těles nabídly další zefektivnění jak kódovacích a dekódovacích algoritmů, tak konstrukce kodérů a dekodérů. Zároveň dosahují nejlepší možné schopnosti opravovat chyby, jaké mohou blokové kódy o dané délce zprávy a kódového slova dosahovat. Efektivnější realizace má ovšem za následek výrazně složitější matematický aparát. V práci jsme uvedli dva typy konstrukce, dva způsoby kódování a základní princip dekódovacích algoritmů při použití Reed-Solomonových kódů. Na závěr kapitoly jsme uvedli konkrétní praktické aplikace Reed-Solomonových kódů a některé z nich rozebrali trochu více do detailu. Jednalo se především o standard vydaný pro komunikaci s vesmírnými plavidly, standard CIRC pro audio CD a přiblíženo bylo také opravování chyb u čárových QR-kódů.

Samoopravné kódy obecně, ale i samotné Reed-Solomonovy kódy nabízí spoustu prostoru k dalšímu zkoumání. Pokrytí celé problematiky by vyžadovalo mnohem větší rozsah práce. Konkrétně by bylo možné se zabývat dalšími různými dekódovacími algoritmy Reed-Solomonových kódů, například využitím Euklidova algoritmu pro opravu nečitelných symbolů. Dále je možnost více prozkoumat implemetaci kodérů a dekodérů, ať už hardwarovou či softwarovou. Výzkum Reed-Solomonových kódů navíc stále probíhá a v řadách vědců se intenzivně pracuje na jejich zdokonalování a optimalizaci kódovacích a dekódovacích algoritmů. Novější přístupy ke konstrukci silných a efektivních Reed-Solomonových kódů využívají například tzv. algebraickou geometrii nebo třeba celulární automaty.

Literatura

[1] GALLIAN, Joseph, A. *Contemporary abstract algebra*. 5th ed. Boston: Houghton Mifflin, 2002. ISBN 978-0618122141.

[2] REED, Irving, S. - CHEN, Xuemin. *Error-control coding for data networks*. Boston: Kluwer Academic Publishers, 1999. ISBN 0-7923-8528-4.

[3] MACWILLIAMS, Florence, Jessie. - SLOANE, Neil. *The theory of error correcting codes*. New York: sole distributors for the U.S.A. and Canada, Elsevier/North-Holland, 1977. ISBN 0-444-85009-0.

[4] *Create Galois field array.*[online]. USA: MathWorks inc., 2018 [cit. 2018-03-18]. Dostupné z: https://www.mathworks.com/help/comm/ref/gf.html.

[5] HILL, Raymond. *A first course in coding theory*. New York: Oxford University Press, 1986. ISBN 0-19-853803-0.

[6] SINGLETON, Richard. *Maximum distance q-nary codes*. IEEE Transactions on Information Theory [online]. 1964, 10(2), 116-118 [cit. 2018-03-25]. DOI: 10.1109/TIT.1964.1053661. ISSN 0018-9448. Dostupné z: http://ieeexplore.ieee.org/document/1053661/.

[7] Singleton bound. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, Poslední změna 28.09.2016 [cit. 2018-03-18]. Dostupné z: https://en.wikipedia.org/wiki/Singleton_bound.

[8] DOSTÁL, Zdeněk. - VONDRÁK, Vít. *Lineární algebra* [elektronická skripta]. Ostrava. c2012. [cit. 2018-03-18]. Dostupné z: http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/linearni_algebra.pdf.

[9] PRANGE, Eugene. *Cyclic Error-Correcting Codes on Two Symbols*. AFCRC-TN-57-103, Air Force Cambridge Research Center, Bedford, Mass., 1957.

[10] REED, Irving, S. - SOLOMON, Gustave. *Polynomial codes over certain finite fields*, M.I.T Lincoln Laboratory Group Report 47.23, 1958.

[11] GORENSTEIN, Daniel. - ZIERLER, Neal. *A class of error correcting codes in p^m symbols*, Journal of the Society of Industrial and Applied Mathematics, Vol. 9, pp.207-214, 1961.

[12] PETERSON, Wesley. *Encoding and error-correction procedures for the Bose-Chaudhuri codes*. IEEE Transactions on Information Theory [online]. 1960, 6(4), 459-470 [cit. 2018-04-19]. DOI: 10.1109/TIT.1960.1057586. ISSN 0018-9448. Dostupné z: http://ieeexplore.ieee.org/document/1057586/.

[13] MASSEY, James. *Shift-register synthesis and BCH decoding.* IEEE Transactions on Information Theory [online]. 1969, 15(1), 122-127 [cit. 2018-04-19]. DOI: 10.1109/TIT.1969.1054260. ISSN 0018-9448. Dostupné z: http://ieeexplore.ieee.org/document/1054260/.

[14] Chien Search. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, Poslední změna 01.01.2017 [cit. 2018-04-08]. Dostupné z: https://en.wikipedia.org/wiki/Chien_search.

[15] FORNEY, George. *On decoding BCH codes.* IEEE Transactions on Information Theory [online]. 1965, 11(4), 549-557 [cit. 2018-04-19]. DOI: 10.1109/TIT.1965.1053825. ISSN 0018-9448. Dostupné z: http://ieeexplore.ieee.org/document/1053825/.

[16] CCSDS.org [online]. CCSDS/ASRC Federal Technical Services, c2017 [cit. 2018-03-25]. Dostupné z: https://public.ccsds.org.

[17] *CCSDS recommendation for telemetry channel coding.*[online]. Tokyo, Japan: CCSDS., 1984 [cit. 2018-03-18]. Dostupné z: https://public.ccsds.org/Pubs/101x0b1s.pdf.

[18] *CCSDS recommended standard for TM synchronization and channel coding.*[online]. Washington, DC, USA: CCSDS., 2017 [cit. 2018-03-18]. Dostupné z: https://public.ccsds.org/Pubs/131x0b3e1.pdf.

[19] QR Code. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, Poslední změna 17.04.2018 [cit. 2018-04-17]. Dostupné z: https://en.wikipedia.org/wiki/QR_code.